

CBS 1.5.4 Terrain Translator  
Technical Report

Tobi L.S. Sellekaerts

November 1997

Prepared for:

7th Army Training Command (7th ATC)  
Grafenwoehr, Germany

Terrain Modeling Project Office (TMPO)  
Washington, D.C.

Under Contract:  
DAJA 22-96-D0069

Prepared by:

Logicon, RDA  
Grafenwoehr, Germany

## Table of Contents

<b>1.0 Previous CBS terrain</b>	<b>3</b>
<b>2.0 Changes to version 1.5.4</b>	<b>3</b>
<b>3.0 File format specifications</b>	<b>4</b>
<b>4.0 Source data</b>	<b>4</b>
<b>5.0 Running the translator</b>	<b>5</b>
<b>6.0 Potential other projects</b>	<b>8</b>
<b>Appendix A: CBS terrain file formats</b>	<b>9</b>
A.1 <name>_roads1.txt, <name>_roads2.txt, <name>_roads3.txt	9
A.2 <name>_strms1.txt, <name>_strms2.txt, <name>_strms3.txt	10
A.3 <name>_veg.txt	11
A.4 <name>_urban.txt	12
A.5 <name>_traf.txt	12
A.6 <name>_tin_low.trn	12
A.7 <name>_bridgs.txt	13
A.8 <name>name.trn	14
A.9 Other necessary files	18
<b>Appendix B: Source code for the translator</b>	<b>19</b>
B.1 Program overview	19
B.2 dcwcbs154.aml	21
B.3 arccbs153.aml	30
B.4 154hydtr.aml	42
B.5 Transportation	44
B.5.1 154trans.aml	44
B.5.2 154trans.perl	45
B.6 Hydrology - linear features	48
B.6.1 154hydro.aml	48
B.6.2 154hydro.perl	49
B.7 Vegetation	51
B.7.1 154veg.aml	51
B.7.2 154veg.perl	52
B.8 Urban	54
B.8.1 154urban.aml	54
B.8.2 154urban.perl	55
B.9 Trafficability	57
B.9.1 154traf.aml	57
B.9.2 154traf.perl	59
B.10 Tins	61
B.10.1 154tins.aml	61
B.10.2 154tins.perl	69
B.10.3 154tins2.perl	70
B.11 Bridges	73
B.11.1 154bridges.aml	73
B.11.2 154bridges1.perl	75
B.11.3 154bridges2.perl	78
B.11.4 154bridges3.perl	80
B.11.5 154bridges4.perl	93
B.12 154others.perl	93
<b>Appendix C: Points of contact</b>	<b>95</b>

In November 1997, TerraSim completed the CBS 1.5.4 terrain exporter. We can now rapidly create the vector data portion of CBS version 1.5.4 terrain; we are not able to create or modify the information on the laser disk. This paper will describe the new terrain, discuss the way it differs from that of previous versions, describe the file formats used by CBS 1.5.4 terrain files, and include the source code of the programs that make up the translator.

## 1.0 Previous CBS terrain

Versions of CBS 1.5.3 and previous used a hex based terrain structure. Hexagons were 2km across. Roads traveled from hex center to hex center; rivers traveled around the edges of hexes. Areal features were defined by hexagon also - urban areas, lakes and oceans, roughness (based on the variation of elevation values found in the hex), and elevation average were all defined by hex area. This portion of the terrain was referred to as the vector portion. TerraSim has written a translator to rapidly create the vector portion of 1.5.3 terrain. Features included in previous versions of CBS include:

- Roads - 3 different sizes
- Rivers - 3 different sizes
- Bridges - Existence only
- Urban areas - 3 different categories
- Trafficability - 5 different categories
- Elevation - Average elevation of each hex
- Roughness - Standard deviation of elevation within each hex
- Land Cover (vegetation) - not included

All CBS terrain, to include 1.5.4, includes another portion called the laser disk. On the disk are maps of the playbox at three different scales:

- 1:250,000
- 1:500,000
- 1:1,000,000

The maps are in a raster format and are rectified - geographically linked to their location on the earth. The raster maps are shown behind the vector information. In the game one can zoom in and out, switching between maps on the disk as necessary, and pan around the playbox. One laser disk is needed for each CBS workstation. Laser disks are made and copied at the National Simulation Center in Leavenworth, Kansas, and take approximately 6 months to create.

CBS exercises can be conducted without the laser disk portion, but the visual aspect of the game is significantly lacking. (BBS has migrated away from laser disks to ADRG, but CBS has no such plans according to the NSC.) To supplement the terrain, a large number of bulky maps would be needed at each workstation. A translator can create terrain where none yet exists, but games played on these terrains must be played without the laser disks. Where terrain currently exists, the translator can be used to update, extend, or otherwise improve the terrain.

## 2.0 Changes to version 1.5.4

In 1997, 1.5.4 replaced 1.5.3. Almost every aspect of CBS vector terrain changed with the new version. Significant differences include:

- There are no more hexagons!
- The paths of rivers, roads, and urban areas are no longer tied to hexagons; they are more accurately represented by vectors following their actual paths.
- Topography is represented by Triangular Irregular Network (TIN) structures, enabling line of sight calculations and more faithful terrain representation.
- Vegetation / land cover is a newly included feature.
- Trafficability regions are no longer defined by hexagons; they are irregular shapes and sizes.
- More information is included about bridges, including Military Load Class (MLC).
- Terrain data is imported to CBS via multiple input files instead of just one.

As with 1.5.3, the translator for 1.5.4 can only create the overlying vector terrain. Features included in CBS 1.5.4:

- Roads - 3 different sizes
- Rivers - 3 different sizes

- Bridges - different categories
- Urban areas - different categories
- Trafficability - different categories
- Elevation - TINs
- Roughness - not included
- Land Cover (vegetation) - different categories

### 3.0 File Format Specifications

A complete CBS 1.5.4 terrain consists of the following suite of files (<name> should be replaced by the whatever name you have assigned your terrain, i.e. 'france'):

- <name>**\_roads1.txt, <name>**\_roads2.txt, <name>**\_roads3.txt**** - roads, split up by size**
- <name>**\_strms1.txt, <name>**\_strms2.txt, <name>**\_strms3.txt**** - rivers, split up by size**
- <name>**\_bridgs.txt** - bridges
- <name>**\_traf.txt** - trafficability regions
- <name>**\_veg.txt** - vegetation / land cover areas
- <name>**\_urban.txt** - urban areas
- <name>**\_tin\_low.trn** - a low resolution tin
- <name>**.trn** - a properly formatted CBS 1.5.3 file (seems that the fields can be empty)
- <name>**rads.dat, <name>rad{0,1,2}.dat** - files with unknown functions, but constant formats so they can be duplicated
- <name>**lines.dat, <name>line{0,1,2}.dat** - files with unknown functions, but constant formats so they can be duplicated
- <name>**parms.dat, <name>parm{0,1,2}.dat** - files that come with and are specific to each laser disk, describing the display parameters for the digital maps on the disk

For the terrain build called lantica, you would have the following file names:

atlantis_traf.txt	atlantisline1.dat
atlantisparm1.dat	atlantisrad1.dat
atlantis.trn	atlantis_roads3.txt
atlantis_urban.txt	atlantisline2.dat
atlantisparm2.dat	atlantisrad2.dat
atlantis_bridgs.txt	atlantis_strms1.txt
atlantis_veg.txt	atlantislines.dat
atlantisparms.dat	atlantisrads.dat
atlantis_roads1.txt	atlantis_strms2.txt
atlantisline0.dat	atlantismap.dat
atlantisrad0.dat	atlantis_strms3.txt
atlantis_roads2.txt	atlantisparm0.dat

Please see Appendix A for descriptions of the actual file formats.

### 4.0 Source Data

As far as file formats are concerned, 1.5.4 terrain bears absolutely no resemblance to 1.5.3 - they might as well be different simulations. Fortunately, the similarity in scale means that we can still use DCW and DTED to build the terrain. Unfortunately, 1.5.4 terrain requires more information than DCW and DTED provide.

<u>Data Needed</u>	<u>Source</u>
Transportation	Digital Chart of the World
Hydrography	Digital Chart of the World
Urban Areas	Digital Chart of the World
Topography for TIN	DTED
Topography where DTED is not available	Digital Chart of the World
Trafficability	DTED
Vegetation	Tactical Pilotage Charts - ADRGs
Bridges	TTADB (future feature)

Unlike the other features, vegetation information has to be extracted from a non-vector source. This single aspect will make the creation of 1.5.4 terrain much more labor intensive than 1.5.3 terrain. It won't necessarily take significantly longer, but much work will have to be done while the translator is running (instead of simply starting it and feeding in disks) extracting vegetation data. Where ADRGs of TPCs are available, I will automatically extract vegetation information from them. If ADRGs are not available, TPCs or ONCs will have to be scanned and rectified first. If we have sufficient imagery coverage, it will be classified to create the necessary vegetation data. Since CBS playboxes can be as large as 100 one degree cells, this is a significant amount of labor.

A future version of the translator may extract exact bridge information from TTADB; currently the exporter classifies all bridges as Military Load Class of 100 (as do all the sample files from NSC). I will write this version if CBS truly uses the MLC bridge classifications; I am not sure if this is the case.

## 5.0 Running the Translator

Here is my advice for actually executing the translator on your machine. The programs were all written for Arc/Info version 7.0.4 and have not been tested on any other version of Arc/Info.

**Please note:** The translator is rather friendly; however, it assumes that the user has a basic working knowledge of Arc/Info. For the vegetation section, you will have to create a vegetation coverage from scratch by digitizing or classifying imagery. The translator is not for the Arc/Info faint of heart.

**Special note about speed of execution:** I run this AML on a Sun Sparc20 Workstation running Solaris 2.5.1. My machine has 256 megabytes of RAM and two 4 gigabyte hard drives. I would suggest a full 4 gigabytes of space to create a CBS playbox for the machine used for the TIN generation.

### Preparation:

1. Ensure that you have a TIN and GRID module as part of your Arc/Info.
2. Ensure that you have a perl compiler accessed in /usr/local/bin.
3. Create a directory that will hold all the files and programs associated with CBS terrain builds. I have a directory called cbs154. Create a subdirectory called aml and put in it all of the programs listed above (amls and perl scripts). Add a line to your .arc file specifying the aml directory as an &AMLPATH. (The programs are all called by dcwcbs154.aml with the &run command, so if they are in your atool directory the program will not find them.)
4. Create a subdirectory of your cbs154 directory called exports. The main program will create an subdirectory here named by the operator, and place all the finished CBS files in it.
5. Create a subdirectory of your cbs154 directory called working (or france or whatever you would like; you will actually be running most of the translator from this directory.)
6. Create a subdirectory of your cbs154 directory called exports.
7. Place the coverages world\_map and world\_tile, and the file longlength.txt in the working directory.
8. If /cdrom/cdrom0 does not point to your CD-ROM drive, set up a link so it does.

Here is my directory structure when I'm ready to begin:

```
disk2
    cbs154
        amls
            dcwcbs154.aml
            154hydtr.aml
            154trans.aml
            154trans.perl
            154hydro.aml
            154hydro.perl
            154bridges.aml
            154bridges1.perl
            154bridges2.perl
```

```

154bridges3.perl
154bridges4.perl
154urban.aml
154urban.perl
154others.perl
arccbs153.aml
154tins.aml
154tins.perl
154traf.aml
154traf.perl
154veg.aml
154veg.perl
working
info
world_tile
world_map
tilelist.txt
exports

```

### **Running the translator:**

The translator is set up so that you can optimize speed if you have the workstations to spare. There are three different parts to the translator that have to be launched by the user: dcwcbs154.aml, 154tins.aml, and 154veg.aml. The first two programs only entail starting the programs and inserting CD-ROMs. The third necessitates some work before starting it. If you have three workstations you can use, I would recommend starting dcwcbs154.aml on Workstation A, then launch 154tins.aml on Workstation B, and finally construct the data on Workstation C before launching 154veg.aml. You would run all these programs remotely from a single monitor so you would know when to go to the other workstations and change CD-ROMs.

**1. Main Program** On one workstation (the one where you set up the directories above; we'll call it Workstation A), start dcwcbs154.aml from your working workspace by typing &r dcwcbs154 at the Arc: prompt. Stay at that workstation until it begins to import data from the DCW disks; periodically you will have to insert more DCW CD-ROMs. Don't be surprised if it prompts you to insert DCW disks that you didn't think you needed. After reading data off of the disks and putting some coverages together, the program will ask you about the existence of hexagon coverages. After you answer the question about hex size, the program needs no more attention and will run to the finish.

### **2. Tin and Trafficability**

Many playboxes will have incomplete DTED coverage - do not worry, your TIN won't be flat in those areas. DCW includes very coarse elevation information. The program will figure out which areas are not covered by DTED and extract elevation information from DCW for those areas. The source data will be coarser but won't be flat. You will not be able to see differences in the TIN between areas with DTED coverage and areas without.

**2a. If you have a second workstation and speed is important:** On another workstation (we'll call it Workstation B), create the same directory structure above. You will only need 154tins.aml and 154tins.perl in your AML directory. You won't need world\_tile, world\_map, or the longlength.txt file. At this point, the program you already launched from Workstation A will have created three coverages you need for this aml. Copy the coverages xtiles, xhy, and xoceanx from the working directory on Workstation A to the working directory on Workstation B. (If xoceanx and xhy haven't been created yet, go on to step 3 and come back later to check.) From the working directory on Workstation B start 154tins.aml by typing &r 154tins from the Arc: prompt. Periodically you will have to come back to Workstation B and insert DTED disks. Once you have inserted all the disks that it needs, however, the program needs no more attention and will run to the finish. You'll know that you have finished inserting DTED disks when you answer Y to the prompt "Is this all the available DTED over this area?". Don't be surprised if it prompts you to insert DTED disks for areas that are slightly outside your playbox - the program needs extra data around the edges to ensure accuracy at the edges. The

program will not automatically copy the output file to your export directory; you will have to do it by hand. The output file will be called <name>\_tin\_low.trn

**2b. If you don't have a second workstation:** Once dcwcbs154.aml has finished running on Workstation A, launch 154tins.aml from the working directory by typing &r 154tins from the Arc: prompt. You have to wait for dcwcbs154.aml to finish so that there are no CD-ROM conflicts and so you aren't running two programs simultaneously in the same workspace. All the coverages and programs that you need for this AML are already where they need to be. Periodically you will have to insert more DTED disks. Once you have inserted all the disks that it needs, however, the program needs no more attention and will run to the finish. You'll know that you have finished inserting DTED disks when you answer Y to the prompt "Is this all the available DTED over this area?". Don't be surprised if the program prompts you to insert DTED disks for areas that are slightly outside your playbox - the program needs extra data *around* the edges to ensure accuracy at the edges. The program will not automatically copy the output file to your export directory; you will have to do it by hand. The output file will be called xxfinal\_tin.trn; move and rename it to <name>\_tin\_low.trn.

### 3. Vegetation

DCW has no vegetation information outside of CONUS, so you have to create the vegetation information yourself. Source data for the original CBS playboxes is at 1:500,000 scale. DCW is at 1:1,000,000 scale. What source you use to create the vegetation coverage will depend on what sources you have available, how much time you have, and your computing resources. You can create the vegetation coverage however you would like (i.e. digitizing paper maps or ADRG, automatically extracting the vegetated areas off of paper maps or ADRG, or classifying imagery). The fastest method I've found is to:

- import the ADRG into Imagine
- export it as a TIFF
- use Adobe Photoshop to open the TIFF
- lift the green areas out being careful not to change the extents or resolution of the TIFF
- use a filter to get rid of holes and dust
- change the TIFF to a bitmap and save it
- convert the TIFF to an Arc/Info GRID
- use GRIDPOLY to convert the image to polygons
- project the polygon coverages into Lambert (be sure to do this before generalization)
- generalize to eliminate the 'stair-step' effect (generalize distance depends on the scale of your ADRG and your latitude)
- use MAPJOIN to combine adjacent map sheets
- build and check for islands and common boundaries

#### Vegetation coverage specifications:

- All vegetation must be in one coverage called xvchg.
- The coverage must be clean with polygon topology.
- The coverage must be in the Lambert Conformal Conic projection. The parameters for the Lambert projection are decided automatically by the dcwcbs154 program. You need to go to the workspace running dcwcbs154.aml and do a describe on any one of these coverages to get the complete Lambert projection description: xoceanx, xhydrox, xtransx, etc. Do not use xhydroxg, xoceanxg, xtransxg, etc. as these are not in the Lambert projection. (If you do a describe on a coverage and it doesn't specifically say Lambert projection, then you've not chosen wisely.)
- The coverage cannot have islands. CBS does not support them; use your best judgement when digitizing.
- Currently all vegetation types are thrown together as a generic vegetation category; when we talk about vegetation in CBS we are really talking about forested areas only. Since all polygons represent vegetation, no two polygons should share a common boundary. (Why would there be a boundary between them when they represent the same thing?) Again, use your best judgement when digitizing.
- The coverage doesn't need any special attributes. Just be sure that it doesn't have any islands and that it is clean with polygon topology.

- The coverage should cover your entire playbox land area but don't worry if the extents aren't the same - there won't be any vegetation over ocean areas which could comprise large sections of the playbox.

**3a. If you have a third workstation and speed is important:** On another workstation (we'll call it Workstation C), create the same directory structure above. You will only need 154veg.aml and 154veg.perl in your AML directory. You won't need world\_tile, world\_map, or the longlength.txt file. Create the vegetation coverage with the above specifications. Launch 154veg.aml from the working directory by typing &r 154veg from the Arc: prompt. The program will not automatically copy the output file to your export directory; you will have to do it by hand. The output file will be called xxvegout1.txt; move and rename it to <name>\_veg.txt

**3b. If you don't have a third workstation:** Once 154tins.aml has finished running on Workstation A, create the vegetation coverage using the above specifications. You have to wait for 154tins.aml to finish so that there are no CD-ROM conflicts and so you aren't running two programs simultaneously in the same workspace. Launch 154veg.aml from the working directory by typing &r 154veg from the Arc: prompt. The program will not automatically copy the output file to your export directory; you will have to do it by hand. The output file will be called xxvegout1.txt; move and rename it to <name>\_veg.txt

Once all the export files have been created, the best way to get them to the CBS machine is to FTP them there. The CBS technicians will do the rest.

## 6.0 Potential Other Projects

If an existing CBS terrain simply needed to be edited, a program could be written to import the files of interest into Arc/Info, edit as necessary, and export them back out to CBS. If the coverage of interest is not salvageable, A variation of the current translator could be written that allows you to make only the files you needed, i.e. if you wanted to make a new roads file without waiting for the program to make all the other files.

## Appendix A

### Format descriptions for CBS 1.5.4 terrain files

Some of these descriptions are taken from the sparse documentation found at the beginning of the sample files I downloaded from the NSC. Each feature has its own file(s) and feature type by number, and is written to the file as a set of ASCII records. All numbers are real numbers and are recorded to six decimal places. There is an additional END following the last entry in the file. The format of each text file is very strict and varies somewhat from file to file. Version 2 of this technical report will include the exact spacing requirements for each file. All sample files shown are from the Central African Republic sample terrain I have created, with a *name* of 'carep'. The following definitions apply to the terms used in the descriptions:

- Type: or Inside\_Value - a 3 digit number specifying which *kind* of feature this particular entry is (i.e. is this particular urban area a city or a town?). The first digit is the same as the feature type number. The other two digits vary by file.
- No\_points - The total number of points in the feature. For polygon features, the endpoint is repeated - an urban polygon with 7 points defining its shape would actually have 8 points by CBS standards.
- <Feature>\_id - a six digit sequential counter of each entry in the file. The first digit is the same as the feature type number. The next six digits count sequentially.
- Length(km) - length of this linear feature in kilometers
- Perimeter(km) - perimeter of this areal feature in kilometers
- Area(km<sup>2</sup>) - area of this areal feature in square kilometers
- Long(deg) - x-coordinate of this point in decimal degrees
- Lat(deg) - y-coordinate of this point in decimal degrees
- Seg\_Length(km) - distance from the current point to the following point in kilometers

Each file has an additional END placed as an end of file marker. For example, here is the last road of carep\_roads3.txt:

```
103 2 2 111091 0 0 0 0  
0.000004 0.0 0.0 0.0 0.0  
13.695900 11.391090 0.002062  
13.696140 11.395080 0.  
END  
END
```

#### A.1 <name>\_roads1.txt, <name>\_roads2.txt, <name>\_roads3.txt - roads, split up by size

Roads are feature type #1. The largest roads go in **roads3**; the smallest in **roads1**. A road with N points occupies N + 3 records:

Record 1:	Type: No_Points	2	Road_ID	0	0	0	0
Record 2:	Length(km)	0.0	0.0	0.0	0.0		
Record 3:	Long(deg)		Lat(deg)		Seg_Length(km)		
Record 4:	Long(deg)		Lat(deg)		Seg_Length(km)		
Record 5:	Long(deg)		Lat(deg)		Seg_Length(km)		
.	.	.	.	.	.	.	
.	.	.	.	.	.	.	
Record I+2:	Long(deg)		Lat(deg)		Seg_Length(km)		
Record I+3:	Long(deg)		Lat(deg)		Seg_Length(km)		
.	.	.	.	.	.	.	
.	.	.	.	.	.	.	
Record N+2:	Long(deg)		Lat(deg)		0.		
Record N+3:	END						

Entries in <name>\_roads1.txt are of Type: 101. Entries in <name>\_roads2.txt are of Type: 102. Entries in <name>\_roads3.txt are of Type: 103. For the two features with multiple files (roads and rivers), the Road\_ID and River\_ID count sequentially through all three files. Here is a subset of the file carep\_roads3.txt:

```

103 9 2 110146 0 0 0 0
  0.000000 0.0 0.0 0.0 0.0
12.774210 11.844780 0.003077
12.812250 11.846370 0.001339
12.839130 11.850950 0.002596
12.880250 11.851610 0.004161
12.895910 11.849300 0.005113
12.902090 11.849160 0.000205
12.917410 11.848830 0.005181
12.924180 11.846510 0.004026
12.939170 11.846900 0.

END
103 2 2 110147 0 0 0 0
  0.000000 0.0 0.0 0.0 0.0
13.156030 11.850010 0.001033
13.165130 11.841790 0.

END

```

Notice that the Road\_ID is 110146. The first entry in carep\_roads1.txt would have a Road\_ID of 100001. We recorded 10,145 roads before starting to write to carep\_roads3.txt.

#### A.2 <name>\_strms1.txt, <name>\_strms2.txt, <name>\_strms3.txt - rivers, split up by size

Rivers are feature type #2. The largest rivers go in **strms3**; the smallest in **strms1**. A river with N points occupies N + 3 records:

```

Record 1: Type: No_Points 2 River_ID 0 0 0 0
Record 2: Length(km) 0.0 0.0 0.0 0.0
Record 3: Long(deg) Lat(deg) Seg_Length(km)
Record 4: Long(deg) Lat(deg) Seg_Length(km)
Record 5: Long(deg) Lat(deg) Seg_Length(km)

.
.
.

Record I+2: Long(deg) Lat(deg) Seg_Length(km)
Record I+3: Long(deg) Lat(deg) Seg_Length(km)

.
.

Record N+2: Long(deg) Lat(deg) 0.
Record N+3: END

```

Entries in <name>\_strms1.txt are of Type: 201. Entries in <name>\_strms2.txt are of Type: 202. Entries in <name>\_strms3.txt are of Type: 203. Again, as with roads, the River\_ID counts sequentially through all three files. Here is a subset of the file carep\_strms3.txt:

```

203 5 2 202249 0 0 0 0
  0.000044 0.0 0.0 0.0 0.0
13.205020 11.977310 0.009494
13.208490 11.978400 0.001983
13.214570 11.981370 0.001043
13.235870 11.992370 0.002555
13.242390 12.000000 0.

END
203 4 2 202250 0 0 0 0
  0.000016 0.0 0.0 0.0 0.0
19.871250 11.985790 0.007015
19.871650 11.990390 0.000306
19.869440 11.996800 0.006719
19.866840 12.000450 0.

END

```

The first entry in carep\_strms1.txt would have a River\_ID of 200001. We recorded 2248 rivers before starting to write to carep\_strms3.txt.

### A.3 <name>\_veg.txt - vegetation / land cover areas

Vegetation is feature type #3. The following file structure definition applies to the vegetation, urban, and trafficability files. In each of these, the polygons are ordered by descending area - the largest polygon by area is first in the list; the smallest is last. A polygon with N vertices occupies N + 3 records:

```

Record 1:    Inside_Value   No_Vertices  2  Poly_ID  Parent_ID Area_Flag 0 0
Record 2:    Perimeter(km)  Area(km2)   0.0  0.0  0.0
Record 3:    Long(deg)     Lat(deg)      Seg_Length(km)
Record 4:    Long(deg)     Lat(deg)      Seg_Length(km)
Record 5:    Long(deg)     Lat(deg)      Seg_Length(km)
.
.
.
Record I+2:  Long(deg)     Lat(deg)      Seg_Length(km)
Record I+3:  Long(deg)     Lat(deg)      Seg_Length(km)
.
.
.
Record N+2:  Long(deg)     Lat(deg)      0.
Record N+3:  END

```

Instead of Type:, areal features have Inside\_Value. The definition is the same. Vegetation polygons have two different inside values: 303 and 399. I have not yet been able to determine the difference; there may be some kind of mobility change from one type to the other. I am hoping and loading terrain and testing it in CBS will help me to determine the difference. I do not know what Parent\_ID or Area\_Flag represent; I have not found a single entry in any of my sample files where either is equal to anything but 0. I think Parent\_ID would apply if the entry were an island in a larger polygon but have yet to find an example proving this either correct or incorrect. Here is the 18th entry from carep\_veg.txt:

```

303 15 2 300018 0 0 0 0
  14.145854      8988.419000  0.0  0.0  0.0
  13.813290      10.724540   0.963000
  13.804440      10.728200   2.539000
  13.781160      10.732750   0.200000
  13.779290      10.735960   0.209000
  13.781140      10.741030   1.221000
  13.792300      10.741060   0.871500
  13.800190      10.747520   0.976500
  13.809010      10.757200   0.159500
  13.810380      10.763650   0.507000
  13.815040      10.762280   0.245000
  13.817390      10.753080   0.086000
  13.818370      10.737900   0.210000
  13.816520      10.731910   0.098000
  13.817460      10.728230   0.461500
  13.813290      10.724540   0.
END

```

### A.4 <name>\_urban.txt - urban areas

Urban areas are feature type #4. For the file structure, please see the definition above for vegetation. The descriptions for Inside\_Value are:

Inside Value	Description
403	City
402	Town
401	Village

Here is the twelfth entry from carep\_urban.txt:

```
403 12 2 400012 0 0 0 0  
15.877189      14477.506000 0.0 0.0 0.0  
26.171830      11.395410   2.013000  
26.153430      11.397890   0.230500  
26.151320      11.398170   0.183000  
26.149920      11.420240   0.879000  
26.141990      11.429890   0.501500  
26.146650      11.435410   0.853500  
26.154570      11.444150   2.451000  
26.176950      11.438170   0.156000  
26.178350      11.435870   0.174000  
26.179750      11.420240   1.003000  
26.170430      11.409660   0.172500  
26.171830      11.395410   0.  
END
```

#### A.5 <name>\_traj.txt - trafficability regions

Trafficability regions are feature type #5. For the file structure, please see the definition above for vegetation. Trafficability polygons tend to be much larger than those for either urban or vegetation. Inside\_Value ranges from 501 to 506. I haven't figured out the differences yet; there is little consistency (i.e. two ocean polygons could be classified as 503 and 506 respectively). Here is the fifty-fourth entry from carep\_traj.txt:

```
501 13 2 500054 0 0 0 0  
6.697657      2490.275750 0.0 0.0 0.0  
14.896460      9.352510   0.030500  
14.896720      9.355210   0.148500  
14.895330      9.358430   0.405500  
14.891630      9.358890   0.106500  
14.892550      9.363030   0.507500  
14.897180      9.363020   0.358500  
14.900430      9.365320   1.064000  
14.910150      9.363920   0.301000  
14.912920      9.361160   0.214000  
14.911060      9.352430   0.963500  
14.902270      9.352900   0.357500  
14.899020      9.351070   0.279000  
14.896460      9.352510   0.  
END
```

#### A.6 <name>\_tin\_low.trn - a low resolution TIN

TINs are feature type #6. This name describes a low resolution TIN specifically. According to JPL, there will be TINs made for each terrain at different resolutions, from a low density TIN to a high density TIN. The first line of most files is a header line, consistently the following:

```
600 0 1 0
```

However, I have found some TIN files without it. It is a flag signaling the relative densify of the TIN to follow? TINs are outlined by the three points that make up each triangle. The unit of measurement is 3 decimal seconds (showing a remarkable affinity to DTED level 1 - you could even call the X and Y coordinates DTED level 1 row and column). Each TIN is written as a set of 4 ASCII records:

```
Record: 1    Triangle_Type 3 1 Triangle_ID Neighbor_1 Neighbor_2 Neighbor_3  
Record: 2 (point1)    X1_coordinate Y1_coordinate Elevation(m)  
Record: 3 (point2)    X2_delta_toX1 Y2_delta_toY1 Elevation(m)  
Record: 4 (point3)    X3_delta_toX1 Y3_delta_toY1 Elevation(m)
```

Triangle\_Type is, in all cases so far, equal to 601. I think this number will change with the density of the described TIN, much as 101, 102, 103 would describe the relative size of a road (602 might be triangles of a medium resolution and 603 a high resolution TIN). Triangle\_ID is a sequential counter as with the other text files; however, in this file it starts counting at 1 and not at 600001 as you would expect. The three Neighbor entries are the Triangle\_ID values of the adjacent triangles:

<u>Item</u>	<u>Segment shared by neighbor</u>
Neighbor_1	Formed by points 1 and 2
Neighbor_2	Formed by points 2 and 3
Neighbor_3	Formed by points 3 and 1

The first triangles described are in the northwest corner of the playbox; the last are in the southeast, traveling south first then east. The way that the geometry of each triangle is described is very important. The locations of points 2 and 3 are described by their distance from point 1. The actual location of what would be x2\_coordinate can be found by adding x2\_delta\_toX1 to x1\_coordinate, and so on:

```

x2_coordinate = x2_delta_toX1 + x1_coordinate
y2_coordinate = y2_delta_toX1 + y1_coordinate

x3_coordinate = x3_delta_toX1 + x1_coordinate
y3_coordinate = y3_delta_toX1 + y1_coordinate

```

Note than even point 3 is defined by its distance from point 1; it would be an easy mistake to think that point 3 would be defined by its distance to point 2 but that is not the case. Here are the first two entries and the header line from carep\_tin\_low.trn:

```

600 0 1 0
601 3 1 3 469 456 465
199800 47808 334
9 -9 338
9 -18 335
601 3 2 3 450 111 494
199890 47835 340
-9 18 341
18 45 337

```

#### A.7 <name>\_bridgs.txt - bridges

Bridges are feature type #7. There is an extensive amount of information listed for each bridge. Road\_ID and Road\_Type describe the road on which this bridge is built - the id and type listed in the roads files for the same terrain. River\_ID and River\_Type similarly describe the river over which the bridge is built. Since this is a point feature, Long(deg) and Lat(deg) describe the center point of each bridge. Military\_Load\_Class (real) and Brdg\_Orientation(deg) are self explanatory. Each bridge/ford is written as a set 4 ASCII records:

```

Record: 1    Brdg_Type   1  2   Brdg_ID Road_ID   Road_Type   River_ID   River_Type
Record: 2    Brdg_Orientation(deg)  0.0  0.0  0.0  0.0
Record: 3    Long(deg)     Lat(deg)   Military_Load_Class(real)
Record: 4    END

```

The example files list all bridges as being of Brdg\_Type 701 and Military\_Load\_Class 10. (Is the MLC being divided by 10 before entry into the file?) The values for Brdg\_Orientation spans the range of the compass from 0 to just less than 180. Bridge direction is taken as heading from southeast through north. For example, a bridge would never have a heading of 270 - it would be 90 instead. Here are the first two entries from carep\_bridgs.txt:

```

701 1 2 700001 100007 1 200043 2
    1.629795 0.0 0.0 0.0 0.0
    24.733000      0.001000     10.000000
END
701 1 2 700002 100008 1 202253 1
    6.747333 0.0 0.0 0.0 0.0
    22.222000      0.975000     10.000000
END

```

#### A.8 <Name>.trn

This is a CBS 1.5.3 terrain file with 36 lines of header information and then a line for each hexagon in the terrain. All numbers in the header are taken to 7 decimal places. All numbers are aligned by the decimal point in the nth column of the page. Here is the header format description:

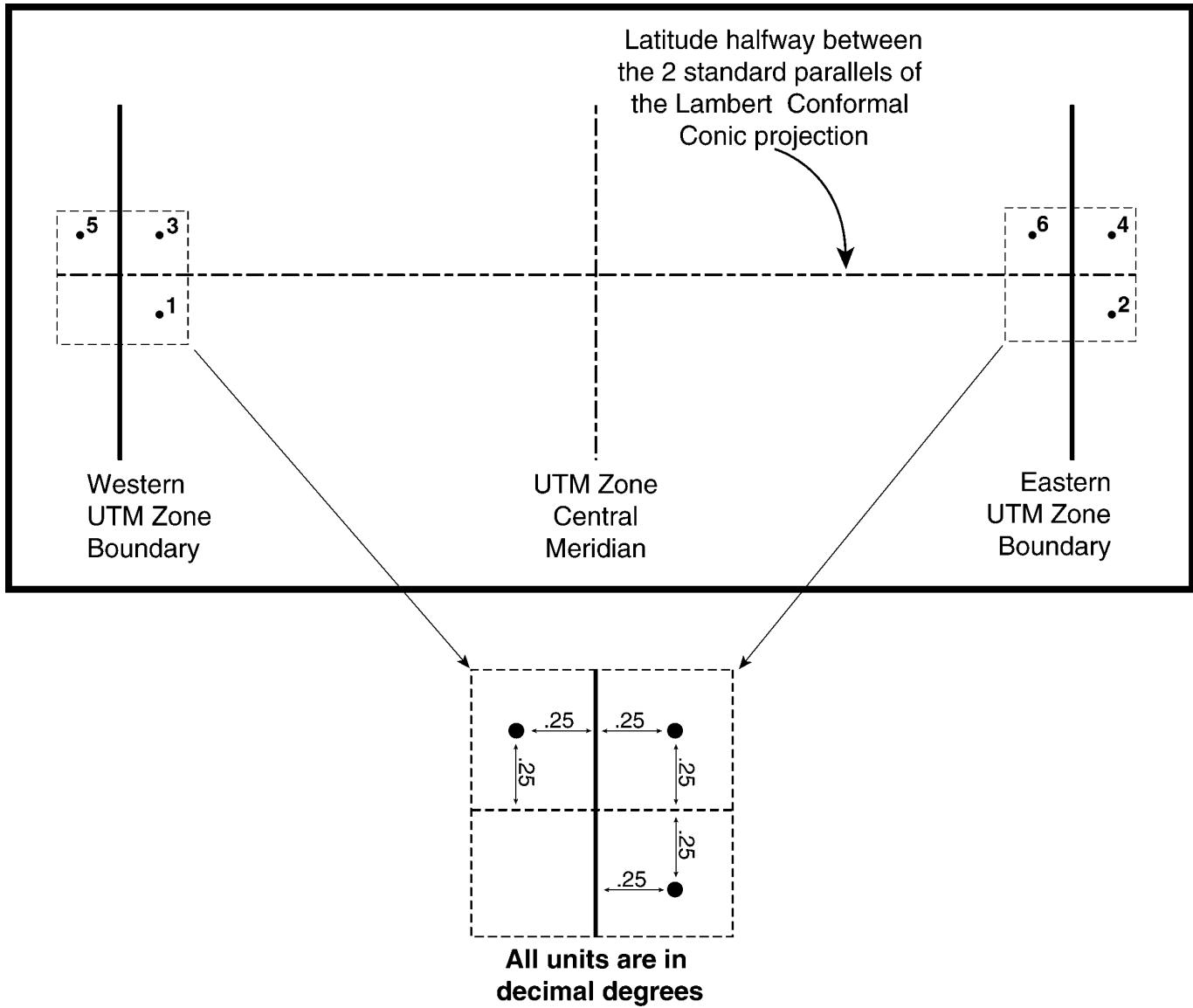
The first 12 numbers are pairs of measurements in meters measured on a UTM projection. The first number is the easting, the second the northing. For example, #1 is the easting and #2 is the northing of a single point, and so on through #12. There are two groups of three points each, centered around the lines of longitude that form the UTM zone boundaries on either side of the UTM zone that falls in the center of the playbox.

I think CBS uses these points to calculate UTM and MGRS coordinates, but I'm not sure why the zone is defined this way instead of traditional zone definition methods.

1. Easting of point #1
2. Northing of point #1
3. Easting of point #2
4. Northing of point #2
5. Easting of point #3
6. Northing of point #3
7. Easting of point #4
8. Northing of point #4
9. Easting of point #5
10. Northing of point #5
11. Easting of point #6
12. Northing of point #6

The hexagons are created in a Lambert Conformal Conic projection. The following three items describe the Lambert projection, and are all in units of decimal degrees.

13. South Standard Parallel
14. North Standard Parallel
15. Central Meridian



The CBS playbox is described by many different units. We've seen UTM, Geographic, and Lambert Conformal Conic so far. Now we introduce a fourth unit of measurement of the playbox - pixels. The pixel unit of measurement is on the Lambert projection. A pixel's size is described by Lambert meters. The record that tells the actual pixel resolution doesn't come until #36. Three of the next five numbers describe the cone used for the Lambert Conformal Conic projection. This information is not usually necessary to describe the Lambert projection. The following numbers are distances measured from the point at the top of the cone. #16 and #20 are so small because they are in units of the earth's radius.

16. The distance from the cone apex to the standard parallel closest to the apex measured in units of the earth's radius value that is being used for calculations. This measurement is called the 'Lambert to pixel conversion constant in the y-direction'.
17. Always 0; don't know what it is used for. I've seen this called the 'orthogonal projection'.
18. The distance from cone apex to northern playbox boundary along the central meridian of the Lambert projection measured in units of pixels
19. Always 0; don't know what it is used for. I've seen this called the 'orthogonal projection'.
20. The same as #16. This measurement is called the 'Lambert to pixel conversion constant in the x-direction' and is the same as #16 because pixels are the same size in both the x and y dimension.

21. The distance from the central meridian to the western edge of the playbox measured on the Lambert projection measured in units of pixels.
22. Half the height of a single hexagon (usually 1500 meters) in pixels
23. The distance from the center of one hexagon to the center of the adjacent (left, right, up, or down, not on the diagonal) hexagon (usually 3000 meters) in pixels
24. Offset from the center of the southwesternmost hex to the southern playbox boundary (shortest distance) on the Lambert projection measured in pixels.
25. Offset from the center of the southwesternmost hex to the western playbox boundary on the Lambert projection measured in pixels.
26. The distance from the north edge of the playbox to the south edge of the playbox, along the central meridian, measured on the Lambert projection in units of pixels.
27. Number of hexes in the database in the east-west direction - number of columns
28. Number of hexes in the database in the north-south direction - number of rows
29. Size of a single hex (length from side to opposite side not corner to opposite corner) in kilometers. For most CBS terrain this value is 3.
30. Each row and column in CBS is numbered from west to east and south to north. This item is the starting number when counting hex columns, the number of the westernmost column. This number is usually 1 and I cannot think of a case in which it would not be.
31. The starting number when counting hex rows, the number of the top row. This number is usually 1.
32. The ending number when counting hex columns, the number of the easternmost column. This is equal to the total number of columns in the database, or the same as item #27.
33. The ending number when counting hex rows, the number of the bottom row. This is equal to the total number of rows in the database, or the same as item #28.
34. Numbers 34 and 35 are always 0. No one seemed to be able to tell me what they were used for, if anything. The typical response when asked was that it was "not used for anything and I don't know what it is..."
35. 0; same as number 34
36. The size of a single square pixel in meters.

Note that with all this information included in the header file, you can access any location in the database in one of 5 ways:

- Latitude and Longitude
- Meters in UTM
- Meters in Lambert Conformal Conic
- Pixels
- Hexagon row and column

The simulation is able to rapidly switch among these various coordinate systems. Here is a sample 36 line header file:

```

433928.4062500
4178355.2500000
962661.0000000
4191088.0000000
434151.3437500
4206093.5000000
961094.5625000
4218854.5000000
390250.8437500
4206565.0000000
917155.1250000
4216487.5000000
        40.0000000
        35.0000000
        32.0000000
31829.3446356
        0.0000000
-39985.9257467
        0.0000000

```

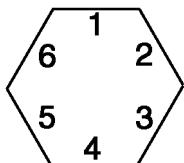
31829.3446356
15751.2439282
7.5000000
12.9903811
0.0000000
0.0000000
2845.5000000
180.0000000
189.0000000
3.0000000
1.0000000
1.0000000
180.0000000
189.0000000
0.0000000
0.0000000
200.0000000

Notice the alignment at the decimal point and the consistent maintenance of 7 decimal places.

Now we come to the body of the CBS 1.5.3 terrain file; there is one line for each hexagon in the database. They are listed first by row then by column. In the description, a line begins with column number 0 (so column 1 is actually the second space on the line).

- Item 1, column 1 - A number that represents the trafficability of the terrain.
  1. water (lake and ocean)
  2. impassable
  3. smooth
  4. gently rolling
  5. mountainous
  6. water based agriculture
- Item 2, column 3 - A number that represents the type of urban area that occupies this hex.
  0. No Settlement
  1. Village
  2. Town
  3. City
- Item 3, column 5 - A number that represents the percentage of vegetation cover within this hex.
  0. none
  1. from 0% to 33%
  2. from 34% to 66%
  3. from 67% to 100%
- Item 4, column 9 - The average elevation, above sea level in kilometers, of this hex.
- Item 5, column 13 - The roughness of this hex (we calculate roughness by finding the standard deviation of the elevation within the hex area) above sea level in kilometers.

The next 18 values are collected together in three groups of six. The first six are road values, the second six are river values, and the third six are bridge values. Each of the six values represent a single hex side. Roads travel from hex center to hex center, intersecting hex sides at 90 degree angles. Rivers flow around the edges of hexes. Bridges occur anywhere a river intersects a road, and the exact middle of a hex edge.



- Item 6, column 15 - A number that represents the type of road crossing hex side 1.
  0. no road
  1. all weather loose surface

- 2. all weather hard surface
  - 3. dual highway
  - Item 7, column 17 - A number that represents the type of road crossing hex side 2.
  - Item 8, column 19 - A number that represents the type of road crossing hex side 3.
  - Item 9, column 21 - A number that represents the type of road crossing hex side 4.
  - Item 10, column 23 - A number that represents the type of road crossing hex side 5.
  - Item 11, column 25 - A number that represents the type of road crossing hex side 6.
  - Item 12, column 27 - A number that represents the type of river found on hex side 1.
    - 0. no river
    - 1. small river
    - 2. medium river
    - 3. large river
    - 4. impassable barrier
  - Item 13, column 29 - A number that represents the type of river found on hex side 2.
  - Item 14, column 31 - A number that represents the type of river found on hex side 3.
  - Item 15, column 33 - A number that represents the type of river found on hex side 4.
  - Item 16, column 35 - A number that represents the type of river found on hex side 5.
  - Item 17, column 37 - A number that represents the type of river found on hex side 6.
  - Item 18, column 39 - A number that represents the type of bridge found on hex side 1.
    - 1. no bridge
    - 2. bridge
  - Item 19, column 41 - A number that represents the type of bridge found on hex side 2.
  - Item 20, column 43 - A number that represents the type of bridge found on hex side 3.
  - Item 21, column 45 - A number that represents the type of bridge found on hex side 4.
  - Item 22, column 47 - A number that represents the type of bridge found on hex side 5.
  - Item 23, column 49 - A number that represents the type of bridge found on hex side 6.
  - Item 24, column 51 -- A pound sign, '#'
  - Item 25, column 55 - The column of the the current hex; increments second
  - Item 26, column 59 - The row of the current hex; increments first

Here are four lines from a sample 1.5.3 file body:

## A.9 Other necessary files

I am unaware of the functions of the following files. However, their format are consistently the same so I can create them for each terrain build. They are all ASCII files.

`<name>rads.dat, <name>rad0.dat, <name>rad1.dat, <name>rad2.dat, <name>lines.dat, <name>line0.dat, <name>line1.dat, <name>line2.dat`

```

999999 99 99 9999999 9999999 9999999 9999999 9999 0 0 Sentinel
999999 99 99 9999999 9999999 9999999 9999999 9999 0 0 Sentinel

```

## Appendix B - Source Code of the Translators

### B.1 Program overview

#### **dcwcbs154.aml**

This is the parent program, managing the flow of the translator. It prompts the user for the name and extents of the playbox, creates a directory for the output files, and imports the necessary data from DCW. Finally, it calls each of the following programs.

#### **arccbs153.aml**

Output - <name>.trn

This AML creates a properly formatted CBS 1.5.3 file, which consists of 36 lines of header information and then as many entries as there are hexes in the terrain. However, some of the <name>.trn files included with the sample 1.5.4 terrains seem to have empty hex entries, leading me to believe that the empty file simply needs to be created with the appropriate header information. So arccbs153.aml varies from the true 1.5.3 exporter in that it leaves all the fields for the hexagon information empty, significantly speeding up the processing time. arccbs153.aml needs no input other than the extents of the playbox. The original program was completed in 1996.

#### **154hydrat.aml**

This AML uses hydrographic and transportation coverages that were created in dcwcbw154.aml. In order to obtain a complete output format, each road must have an association with each river it crosses, and each river must have an association with each road it flows under. To accomplish this, some data processing is necessary before the hydro and trans exporters are executed.

#### **154trans.aml, 154trans.perl**

Output - <name>\_roads1.txt, <name>\_roads2.txt, <name>\_roads3.txt

Imports road transportation information from DCW and converts it to the appropriate CBS format.

#### **154hydro.aml, 154hydro.perl**

Output - <name>\_strms1.txt, <name>\_strms2.txt, <name>\_strms3.txt

Imports river information from DCW and converts it to the appropriate CBS format.

#### **154veg.aml, 154veg.perl**

Output - <name>\_veg.txt

User creates vegetation polygon coverages; this program converts them to the appropriate CBS format. These are the only programs that are not called by dcwcbs154.aml. They should be launched as soon as the vegetation coverages are created.

#### **154urban.aml, 154urban.perl**

Output - <name>\_urban.txt

Imports urban area information from DCW and converts it to the appropriate CBS format.

#### **154traf.aml, 154traf.perl**

Output - <name>\_traf.txt

Uses thinned elevation files created by 154tins.aml to create trafficability regions and convert them to the appropriate CBS format.

#### **154tins.aml, 154tins{2,3}.perl**

Output - <name>\_tin\_low.trn

Imports elevation information from DTED and DCW, thins it, creates a TIN, and converts it to the appropriate CBS format.

#### **154bridges.aml, 154bridges{1,2,3,4}.perl**

Output - <name>\_brdgss.txt

Calculates bridge information based on the intersections of road and river data from DCW and converts it to the appropriate CBS format. A future improvement may be to access TTADB data from the bridge exporter to obtain accurate Military Load Class ratings.

#### **154others.perl**

Output - <name>rads.dat, <name>rad{0,1,2}.dat, <name>lines.dat,  
<name>line{0,1,2}.dat  
Create non-terrain specific files needed by CBS.

The programs are launched in the following order. All programs in the leftmost column are launched by the operator; the rest are launched by other programs.

```
dcwcbs154.aml
    154hydtr.aml
    154trans.aml
        154trans.perl
    154hydro.aml
        154hydro.perl
    154bridges.aml
        154bridges1.perl
        154bridges2.perl
        154bridges3.perl
        154bridges4.perl
    154urban.aml
        154urban.perl
    154others.perl
    arccbs153.aml
154tins.aml
    154tins.perl
    154traf.aml
        154traf.perl
154veg.aml
    154veg.perl
```

## B.2 dcwcbs154.aml

```

/* dcwcbs154.aml
/*
-----*
* Program: dcwcbs154.aml
* Purpose: The structure of the program is as follows:
* First, the user chooses the
* areas he would like to include in his playbox. This can be done
* in two ways:
*   1. Select some boxes from a 5 by 5 degree grid laid over a map of
*      the world.
*   2. Select the corners of the playbox by UTM or lat/long
coordinates.
* Second, the needed coverages are extracted from the DCW CD-ROM.
Third,
* these coverages are merged and interior borders dissolved. Finally, a
* separate AML called arccbs.aml is called and the .trn file created.
Control
/* is returned to this AML at the end to clean up.
/*
-----*
* Called By: none
* Calls Made: arccbs153.aml
*   154trans.aml - formats roads and rivers
*   154hydro.aml - formats linear hydro features
*   154urban.aml - formats urban points and polygons
*   154tins.aml - creates and outputs the TIN file
*   154bridges.aml - creates and output the bridges file
* Also need world_tile and world_map in the same directory
/*
-----*
* Arguments: none
* Globals: .hexsize - the size of the hexes for the .trn file
*   .name - trunk name for the output files
*   .dirname - name of the destination directory for the files
* Locals: too many to list here...
/*
-----*
* Inputs: Digital Chart of the World
* Outputs: CBS 154 format terrain files
/*
-----*
* History: Written in June and July 1995 for CBS 1.5.3 by Tobi
Steinberg.
* Modified in April and May 1996 by Tobi Steinberg.
* Terrain Simulations - CUBIC Inc. - Grafenwoehr, Germany
* Logicon, TSI as of March 1997
* Began modification for use with CBS 154 in May 1997.
* Began working on TIN portion in August 1997 - Tobi Sellekaerts
/*
-----*

&echo &on
&messages &on
&s junk [close -all]

&if [exists dcwcbs154[date -tag].wat -file] &then
  &s delwat [delete dcwcbs154[date -tag].wat]
&watch dcwcbs154[date -tag].wat; &ty [date -VFULL]

&s cdpath /cdrom/cdrom0

```

```

display 9999 3

&do file &list xtilelist.txt xformat.txt          /* Deleteing needed
files
  &if [exists %file% -file] &then &sys rm %file%
&end

&label badlength
&ty Please make sure that the name of your area is less than
&ty five characters long.
&ty
&ty
&s name [response 'What would you like to call this area?' area]
&if [length %name%] gt 5 &then &goto badlength
&s .name %name%

/* Making an outputting directory for the files
&s test .TRUE.
&s n 1
&if [exists %name%0 -directory] &then &do
  &do &while %test%
    &if ^ [exists %name%%n% -directory] &then &do
      &sys mkdir %name%%n%
      &if ^ [exists %name%%n% -directory] &then &do
        &ty Unable to create directory %name%%n%.
        &return
      &end
      &s .dirname %name%%n%
      &s test .FALSE.
    &end
    &else &s n [calc %n% + 1]
  &end
&end
&else &do
  &sys mkdir %name%0
  &s .dirname %name%0
&end

&do cov &list xtiles xtilesutm xclipcov xclipproj ~ /* Deleteing needed
coverages
xtemp xrd xdn xpp xtransx xdn1 xpo xpppt xhy xelevx ~
xlakesx xhydrox xurbanx xforest xhexx xoceanx xtemp
  &if [exists %cov% -cover] &then kill %cov% all
&end

***** ****
/* Deciding what areas of the world are to be imported into ARC
***** ****
/* Variables, files, coverages used:
/*   xtilelist.txt - list of DCW tile names to be used
/*   total_t(n) - the total number of tiles chosen from each CD
/*   xtiles - a coverage containing the selected tiles
/*   method - the way the user would like to choose the area

&ty You may choose your geographical area in one of two ways:
&ty      1. View a map of the world with 5 by 5 degree boxes placed

```

```

&ty      across its surface and choose a group of boxes covering
&ty      your area of interest.
&ty      2. Specify the upper left and lower right corners of your
&ty      area of interest by latitude and longitude or UTM
&ty      coordinates.

&label method
&s method [response 'How would you like to select your area? [1] or [2]' 1]
&if %method% ne 1 and %method% ne 2 &then &do
  &ty Please enter a 1 or a 2.
  &goto method
&end

&if %method% = 1 &then &do
ae
  &label choose
  ec world_tile
  me def
  ef poly
  de poly
  bc world_map 5
  be arc
  draw
    &ty Choose squares to select the area that you would like to
    &ty include in your CBS terrain.
    &ty Please choose a contiguous area and press 9 when finished.
  &label select
  sel many
  &if [show number selected] = 0 &then &do
    &ty Please choose at least one tile.
    &goto select
  &end
  put xtiles
  save

  ec xtiles
  build
  mape xtiles
  draw
  &s decision [response 'Is this the area you wish to use? [y]' y]
  &if %decision% = y &then &do
    ef poly
    sel cd = 1
    &s total_t1 [show number selected]
    sel cd = 2
    &s total_t2 [show number selected]
    sel cd = 3
    &s total_t3 [show number selected]
    sel cd = 4
    &s total_t4 [show number selected]
  &end
  &else &do
    ec world_tile
    kill xtiles
    y
    &goto choose

```

```

  &end
  save
  quit

  tables
    sel xtiles.pat
    sort cd
    unload xtilelist.txt tile_name columnar xformat.txt
    sort xtiles-id
  quit

  &sys rm xformat.txt

  &end

  /* ****
  /* Selecting an upper left and lower right corner.
  /* ****
  /* Coverages, files, and variables used:
  /*      cs - type of coordinate system used to input corner locations
  /*      zone - utm zone the coordinates fall into
  /*      ulx - upper left x coordinate
  /*      uly - upper left y coordinate
  /*      lrx - lower right x coordinate
  /*      lry - lower right y coordinate
  /*      xclipcov - coverage with the playbox extents
  /*      xclipproj - playbox extents coverage projected
  /*      xtemp - temporary coverage used to project another coverage
  /*      hemi - the hemisphere the playbox resides in
  /*      lat, long, lat(n), long(n) - used to determine the geographic
  /*          center of the playbox area chosen

  &if %method% = 2 &then &do
    &s cs [response 'Please select the coordinate system you would like to
use: ~
    Lat/Long [1] or UTM [2]' 1]
    &if %cs% = 2 &then &do
      &ty You have chosen to enter your coordinates from the UTM
projection.
      &ty You may only enter UTM coordinates if your entire playbox falls
      &ty wholly in either the northern or southern hemisphere (it may
not
        &ty cross the equator.)
        &ty Please use coordinates from a WGS84 spheroid and datum.
        &ty The edges of the playbox will still be defined by lines of
latitude and
        &ty longitude (as used by CBS).
        &ty Please enter all values in meters.
        &label zone
        &s zone [response 'Which UTM zone do these coordinates fall into' ]
        &if %zone% = 0 or %zone% gt 60 &then &do
          &ty Please enter a number between 1 and 60.
          &goto zone
        &end
        &s hemi [response 'Is this area in the southern hemisphere' n]
      &end
      &else &do

```

```

&s hemi 0
&ty You have chosen to enter latitude and longitude coordinates.
&ty Please enter coordinates from a WGS84 spheroid and datum.
&ty Please enter all values in decimal degrees. Latitudes in the
&ty southern hemisphere and longitudes in the eastern hemisphere
&ty should be entered as negative values.
&end

&ty You will be entering coordinates for the upper left and
&ty lower right corners of the box only.

&label badulx
&s ulx [response '      The x value of the upper left corner']
&if %cs% = 1 &then
  &if %ulx% gt 90 or %ulx% lt -90 &then &do
    &ty Please enter a number between -90 and 90.
    &goto badulx
  &end

&label baduly
&s uly [response '      The y value of the upper left corner']
&if %cs% = 1 &then
  &if %uly% gt 180 or %uly% lt -180 &then &do
    &ty Please enter a number between -180 and 180.
    &goto baduly
  &end

&label badlrx
&s lrx [response '      The x value of the lower right corner']
&if %cs% = 1 &then
  &if %lrx% gt 90 or %lrx% lt -90 &then &do
    &ty Please enter a number between -90 and 90.
    &goto badlrx
  &end

&label badlry
&s lry [response '      The y value of the lower right corner']
&if %cs% = 1 &then
  &if %lry% gt 180 or %lry% lt -180 &then &do
    &ty Please enter a number between -180 and 180.
    &goto badlry
  &end

create xclipcov          /* Creating a clip coverage using the
tables                  /* coordinates entered above.
sel xclipcov.tic
add
1
%ulx%
%uly%
2
%lrx%
%uly%
3
%lrx%
%lry%
4

```

```

%ulx%
%lry%
~
quit

display 0
ae
ec xclipcov
ef arc
coordinates keyboard
add
2 %ulx% %uly%
1 %lrx% %uly%
1 %lrx% %lry%
1 %ulx% %lry%
2 %ulx% %uly%
~
coordinates mouse
save
quit
build xclipcov poly

/* Since I am using everything in geographic coordinates, I need to
/* projectdefine the lat/long coverage and project the UTM coverage
/* into geographic.

&s longhemi 0

&if %cs% = 1 &then &do
projectdefine cover xclipcov
  projection geographic
  units dd
  spheroid wgs84
  parameters
copy xclipcov xcliproj
&end
&else &do
  project cover xclipcov xcliproj
  input
  projection utm
  units meters
  zone %zone%
  parameters
  output
  projection geographic
  units dd
  spheroid wgs84
  parameters
  end
&if %hemi% = y &then &do
  &describe xcliproj
  &s lat %dsc$ymin% - 90
  &s long %dsc$xmin%
  kill xcliproj all
  project cover xclipcov xcliproj
  input
  projection utm

```

```

units meters
parameters
%long% 0 0
%lat% 0 0
output
projection geographic
units dd
spheroid wgs84
parameters
end
&end
&end

build xcliproj poly
clip world_tile xcliproj xtiles poly

display 0

ae
  ec xtiles
  ef poly
  sel cd = 1
  &s total_t1 [show number selected]
  sel cd = 2
  &s total_t2 [show number selected]
  sel cd = 3
  &s total_t3 [show number selected]
  sel cd = 4
  &s total_t4 [show number selected]
quit

tables
  sel xtiles.pat
  sort cd
  unload xtilelist.txt tile_name columnar xformat.txt
  sort xtiles-id
quit

&sys rm xformat.txt
&end

&s .hexsize [response 'What is your desired hex size? [3000]' 3000]
&s modify [response 'Would you like to modify the coverages after ~
they have been extracted from DCW and combined? 1 = yes, 2 = no [2]' 2]

&label badname
&s blah [response "Do you already have a .trn file? [n]" n]
&if %blah% = y &then &do
  &s trnname [response "What is the name of your .trn file? ~
[%name%.trn]" %.name%.trn]
  &if [exists $trnname% -file] &then &do
    &system cp %trnname% %dir%/%trnname%
    &s 153test .FALSE.
  &end
  &else &do
    &type This file does not exist.
    &goto badname
  &end
  &end
  &end
  &else &s 153test .TRUE.

/*
 ****
/* Extracting the chosen areas and necessary coverages from CD-ROMS
/* ****
/* Variables, files, and coverages used:
/*      tile - tile number/names as read out of the tile list file
/*      tile2 - tile name after backslashed corrected
/*      cover - tile name without any slashes

&ty Extracting the chosen areas and necessary coverages from CD-ROM

&s tilelist [open xtilelist.txt os -read]

&if %total_t1% ne 0 &then &do
  &if ^ [exists %cdpath%/dcw/noamer -directory] &then &do
    &sys eject
    &ty Please insert DCW Disk #1 into the CD-ROM drive
    &pause
  &end
  &do loop1 = 1 &to %total_t1%
    &s tile [read %tilelist% rs]
    &s a [locase [substr %tile% 1 1]]
    &s b [locase [substr %tile% 3 1]]
    &s c [locase [substr %tile% 5 2]]
    &s tile2 %a%/%b%/%c%
    &s cover %a%b%c%
    &do foo &list pp rd dn po hy
      &if [exists x%foo%%cover%pnt -cover] &then kill
    x%foo%%cover%pnt all
      &if [exists %cdpath%/dcw/noamer/%foo%/%tile2% -directory] AND
    ~
      ^ [exists x%foo%%cover% -cover] &then &do
        vpfimport cover %cdpath%/dcw/noamer/%foo%/%tile2%
    x%foo%%cover%
      build x%foo%%cover% line
    &end
    &if ^ [exists xpppt%cover% -cover] AND [exists xpp%cover%pnt
    -cover] &then
      copy xpp%cover%pnt xpppt%cover%
      &if [exists x%foo%%cover%pnt -cover] &then kill
    x%foo%%cover%pnt all
      &end
    &end
  &end
  &if %total_t2% ne 0 &then &do
    &if ^ [exists %cdpath%/dcw/eurnasia -directory] &then &do
      &sys eject
      &ty Please insert DCW Disk #2 into the CD-ROM drive
      &pause
    &end
    &do loop2 = 1 &to %total_t2%
      &s tile [read %tilelist% rs]
      &s a [locase [substr %tile% 1 1]]
```

```

&s b [lowercase [substr %tile% 3 1]]
&s c [lowercase [substr %tile% 5 2]]
&s tile2 %a%/%b%/%c%
&s cover %a%b%c%
&do foo &list pp rd dn po hy
  &if [exists x%foo%%cover%pnt -cover] &then kill
x%foo%%cover%pnt all
  &if [exists %cdpath%/dcw/eurnasia/%foo%/%tile2% -directory]
AND ~
  ^ [exists x%foo%%cover% -cover] &then &do
    vpfimport cover %cdpath%/dcw/eurnasia/%foo%/%tile2%
x%foo%%cover%
  build x%foo%%cover% line
  &end
  &if ^ [exists xpppt%cover% -cover] AND [exists xpp%cover%pnt
-cover] &then
    copy xpp%cover%pnt xpppt%cover%
    &if [exists x%foo%%cover%pnt -cover] &then kill
x%foo%%cover%pnt all
  &end
  &end
&end

&if %total_t3% ne 0 &then &do
  &if ^ [exists %cdpath%/dcw/soamafr -directory] &then &do
    /* &sys eject
      &ty Please insert DCW Disk #3 into the CD-ROM drive
      &pause
    &end
    &do loop3 = 1 &to %total_t3%
      &s tile [read %tilelist% rs]
      &s a [lowercase [substr %tile% 1 1]]
      &s b [lowercase [substr %tile% 3 1]]
      &s c [lowercase [substr %tile% 5 2]]
      &s tile2 %a%/%b%/%c%
      &s cover %a%b%c%
      &do foo &list pp rd dn po hy
        &if [exists x%foo%%cover%pnt -cover] &then kill
x%foo%%cover%pnt all
        &if [exists %cdpath%/dcw/soamafr/%foo%/%tile2% -directory]
AND ~
        ^ [exists x%foo%%cover% -cover] &then &do
          vpfimport cover %cdpath%/dcw/soamafr/%foo%/%tile2%
x%foo%%cover%
          build x%foo%%cover% line
          &end
          &if ^ [exists xpppt%cover% -cover] AND [exists xpp%cover%pnt
-cover] &then
            copy xpp%cover%pnt xpppt%cover%
            &if [exists x%foo%%cover%pnt -cover] &then kill
x%foo%%cover%pnt all
            &end
            &end
          &s junk [close %tilelist%]

&ty
&ty                               Done.

/* ****
/* Appending coverages and dissolving common borders
/* ****
&s total %total_t1% + %total_t2% + %total_t3% + %total_t4%
&if %total% ne 1 &then &do
  &ty Appending coverages and dissolving common borders.

  &s junk [close -all]

  &do covtype &list rd dn pp rd po hy /* Add hy here later when you have
the disks!
  &if %covtype% = rd &then &do
    &s tilelist [open xtilelist.txt os -read]
    /* &s junk [read %tilelist% rs]
    &s total-1 %total% - 1
    &s tile [read %tilelist% rs]
      &s a [lowercase [substr %tile% 1 1]]
      &s b [lowercase [substr %tile% 3 1]]
      &s c [lowercase [substr %tile% 5 2]]
```

```

&sys eject
  &ty Please insert DCW Disk #4 into the CD-ROM drive
  &pause
&end
&do loop4 = 1 &to %total_t4%
  &s tile [read %tilelist% rs]
  &s a [lowercase [substr %tile% 1 1]]
  &s b [lowercase [substr %tile% 3 1]]
  &s c [lowercase [substr %tile% 5 2]]
  &s tile2 %a%/%b%/%c%
  &s cover %a%b%c%
  &do foo &list pp rd dn po hy
    &if [exists x%foo%%cover%pnt -cover] &then kill
x%foo%%cover%pnt all
    &if [exists %cdpath%/dcw/sasaus/%foo%/%tile2% -directory] AND ~
    ^ [exists x%foo%%cover% -cover] &then &do
      vpfimport cover %cdpath%/dcw/sasaus/%foo%/%tile2%
x%foo%%cover%
      build x%foo%%cover% line
      &end
      build x%foo%%cover% line
      &if ^ [exists xpppt%cover% -cover] AND [exists xpp%cover%pnt
-cover] &then
        copy xpp%cover%pnt xpppt%cover%
        &if [exists x%foo%%cover%pnt -cover] &then kill
x%foo%%cover%pnt all
        &end
        &end
      &end
      &s junk [close %tilelist%]

&ty
&ty                               Done.

/* ****
/* Appending coverages and dissolving common borders
/* ****
&s total %total_t1% + %total_t2% + %total_t3% + %total_t4%
&if %total% ne 1 &then &do
  &ty Appending coverages and dissolving common borders.

  &s junk [close -all]

  &do covtype &list rd dn pp rd po hy /* Add hy here later when you have
the disks!
  &if %covtype% = rd &then &do
    &s tilelist [open xtilelist.txt os -read]
    /* &s junk [read %tilelist% rs]
    &s total-1 %total% - 1
    &s tile [read %tilelist% rs]
      &s a [lowercase [substr %tile% 1 1]]
      &s b [lowercase [substr %tile% 3 1]]
      &s c [lowercase [substr %tile% 5 2]]
```

```

    &s cover xrd%a%%b%%c%
    &if [exists xrd -cover] &then kill xrd all
    append xrd line all
        &if [exists %cover% -cover] &then %cover%
        &do loop = 1 &to %total-1%
            &s tile [read %tilelist% rs]
            &s a [locase [substr %tile% 1 1]]
            &s b [locase [substr %tile% 3 1]]
            &s c [locase [substr %tile% 5 2]]
            &s cover xrd%a%%b%%c%
            &if [exists %cover% -cover] &then %cover%
        &end
    ~
    Y
    Y
    &s junk [close %tilelist%]
    &if [exists xcliproj -cover] &then &do
        build xcliproj
        clip xrd xcliproj xtemp line
        kill xrd all
        build xtemp line
        rename xtemp xrd
    &end
    &end
    &else &do
        &s tilelist [open xtilelist.txt os -read]
        /* &s junk [read %tilelist% rs]
        &if [exists xcliproj -cover] &then &do
            mapjoin x%covtype% net features xcliproj
        &end
        &else mapjoin x%covtype% net features
        &do loop = 1 &to %total%
            &s tile [read %tilelist% rs]
            &s a [locase [substr %tile% 1 1]]
            &s b [locase [substr %tile% 3 1]]
            &s c [locase [substr %tile% 5 2]]
            &s cover x%covtype%a%%b%%c%
            &if [exists %cover% -cover] &then %cover%
        &end
    ~
    Y
    Y
    &s junk [close %tilelist%]
    &end
    &end
    /* Now append the point features of the urban coverage
    &s tilelist [open xtilelist.txt os -read]
    /* &s junk [read %tilelist% rs]
    &s total-1 %total% - 1
    &s tile [read %tilelist% rs]
        &s a [locase [substr %tile% 1 1]]
        &s b [locase [substr %tile% 3 1]]
        &s c [locase [substr %tile% 5 2]]
        &s cover xpppt%a%%b%%c%
    &if [exists xpppt -cover] &then kill xpppt all
    append xpppt point all

    &if [exists %cover% -cover] &then %cover%
    &do loop = 1 &to %total-1%
        &s tile [read %tilelist% rs]
        &s a [locase [substr %tile% 1 1]]
        &s b [locase [substr %tile% 3 1]]
        &s c [locase [substr %tile% 5 2]]
        &s cover xpppt%a%%b%%c%
        &if [exists %cover% -cover] &then %cover%
    &end
    ~
    Y
    Y
    &s junk [close %tilelist%]
    &if [exists xcliproj -cover] &then &do
        build xcliproj
        clip xpppt xcliproj xtemp point
        kill xpppt all
        build xtemp point
        rename xtemp xpppt
    &end

    /* rename and dissolve where necessary to remove tile edge boundaries
    rename xrd xtransx
    dissolve xdn xlakesx dnpypotype poly
    dissolve xdn xhydrox #all line
    dissolve xpp xurbanx pppypotype poly
    dissolve xpo xoceanx popytype poly

    /* Deleting polygons in the lakes coverage that aren't
    /* really lakes and rivers in the hydro coverages that
    /* aren't really rivers, also political borders

    ae
        &if [exists xoceanx -cover] &then &do
            ec xoceanx
            ef poly
            sel popytype = 1
            delete
            save
        &end
        ec xlakesx
        ef poly
        sel dnpypotype = 2
        &if [show number selected] ne 0 &then put xdn1
        &if [exists xoceanx -cover] &then &do
            ec xoceanx
            ef poly
            sel all
            &if [show number selected] ne 0 &then &do
                &if [exists xdn1 -cover] &then &do
                    put xdn1
                    Y
                &end
                &else put xdn1
            &end
            ec xhydrox

```

```

ef arc
sel dnltype ne 1
  &if [show number selected] ne 0 &then delete
save
ec xurbanx
ef poly
sel ppytype ne 1
  &if [show number selected] ne 0 &then delete
save
quit

&if [exists xdnl -cover] &then &do
  kill xlakesx all
  rename xdnl xlakesx
&end
&end

&else &do
  &s tilelist [open xtilelist.txt os -read]
  &s tile [read %tilelist% rs]
  &s a [locase [substr %tile% 1 1]]
  &s b [locase [substr %tile% 3 1]]
  &s c [locase [substr %tile% 5 2]]
  copy xrd%a%%b%%c% xtransx
  copy xpp%a%%b%%c% xurbanx
  copy xdn%a%%b%%c% xhydrox
  copy xdn%a%%b%%c% xlakesx
  copy xpo%a%%b%%c% xoceanx

ae
ec xoceanx
ef poly
sel popytype = 1
delete
save
ec xlakesx
ef poly
sel dnpypotype = 1
  &if [show number selected] ne 0 &then put xdnl
sel dnpypotype = 2
  &if [show number selected] ne 0 &then &do
    &if ^ [exists xdnl -cover] &then put xdnl
    &else &do
      put xdnl
      y
    &end
  &end
ec xoceanx
ef poly
sel all
&if [show number selected] ne 0 &then &do
  &if [exists xdnl -cover] &then &do
    put xdnl
    y
  &end
  &else put xdnl

```

---

```

&end
ec xhydrox
ef arc
sel dnltype ne 1
  &if [show number selected] ne 0 &then delete
save
ec xurbanx
ef poly
sel ppytype ne 1
  &if [show number selected] ne 0 &then delete
save
quit

&if [exists xdnl -cover] &then &do
  kill xlakesx all
  rename xdnl xlakesx
&end
&end

/* ****
/* Projecting all necessary coverages from geographic to lcc
/* ****

/* finding the maximum extents of any of the coverages
&s xmin 180
&s xmax -180
&s ymin 90
&s ymax -90
&do cover &list xlakesx xtransx xurbanx xhydrox xpppt
  &describe %cover%
  &if %dsc$xmin% lt %xmin% &then &s xmin %dsc$xmin%
  &if %dsc$xmax% gt %xmax% &then &s xmax %dsc$xmax%
  &if %dsc$ymin% lt %ymin% &then &s ymin %dsc$ymin%
  &if %dsc$ymax% gt %ymax% &then &s ymax %dsc$ymax%
&end
&s .cm [round [calc [%xmin% + %xmax%] / 2]]
&s .ysix [calc [%ymax% - %ymin%] / 6]
&s .sp1 [truncate [calc [%ymax% - %ysix%] + 1]]
&s .sp2 [truncate [calc %ymin% + %ysix%]]

&if [calc %.sp1% - %.sp2%] lt 4 &then &do
  &s .sp1 [calc %.sp1% + 2]
  &s .sp2 [calc %.sp2% - 2]
&end

/* testing to see if the numbers straddle the equator
/* if so, set the one closest to the equator to 0
&if %.sp1% ne 0 AND %.sp2% ne 0 &then &do
  &if [calc [abs %.sp1%] / %.sp1%] ne [calc [abs %.sp2%] / %.sp2%] &then
    &if [abs %.sp1%] gt [abs %.sp2%] &then &do
      &s .sp2 0
    &end
    &else &do
      &s .sp1 0
    &end

```

```

&end
&end

&do foo &list xlakesx xtransx xurbanx xhydrox xpppt

projectdefine cover %foo%
  projection geographic
  units dd
  spheroid wgs84
  datum wgs84
  parameters

project cover %foo% xdnl
  output
  projection lambert
  units meters
  datum wgs84
  spheroid wgs84
  parameters
  %.sp1% 0 0
  %.sp2% 0 0
  %.cm% 0 0
  0 0 0
  5000000
  0
  end
  &if [exists %foo% -cover] &then kill %foo% all
  copy %foo% %foo%
  kill %foo% all
  rename xdnl %foo%
&end

/* Killing the hydro and trans geographic versions; will recreate later
kill xhydroxg all
kill xtransxg all

kill xppptg all
kill xurbanxg all

/* Now doing special things to deal with the pp point coverage
/* Buffer the urban point features, then add to the urban coverage
&if [exists xppptbuf -cover] &then kill xppptbuf all
buffer xpppt xppptbuf # # 1 # point
additem xppptbuf.pat xppptbuf.pat pppytype 4 12 B
tables
  sel xppptbuf.pat
  calc pppytype = 1
quit

build xurbanx
additem xurbanx.pat xurbanx.pat inside 4 5 B # xurbanx-id

update xurbanx xppptbuf xtemp poly
kill xurbanx all
rename xtemp xurbanx
kill xppptbuf all

/* Running moveislands so there are none inside the urban polygons
build xurbanx
copy xurbanx xtemp
copy xurbanx xppptbuf
moveislands xurbanx xtemp xppptbuf
kill xppptbuf all
kill xtemp all

/* In tables, setting the value of xurbanx-id for cross reference later
tables
  sel xurbanx.pat
  calc xurbanx-id = xurbanx# - 1
quit

project cover xurbanx xurbanxg
  output
  projection geographic
  units dd
  spheroid wgs84
  datum wgs84
  parameters
  end

build xurbanxg
build xlakesx

/* *****
/* Giving the user the opportunity to add to or modify the coverages
before
/* passing them off to the subroutines
/* *****

&if %modify% = 1 &then &do
  lc
  &ty Time to modify the coverages
  &ty Press <return> when you are finished.
  &pause
&end

/* *****
/* Now calling external programs which will create all the necessary
/* files for cbs 154 terrain and move them into the destination directory
/* *****
&ty Creating CBS154 terrain files
/* Doing the TIN first so that the user doesn't have to hang around to
/* feed in the DTED disks.
&type Formatting the tin (feature #3)
&r 154tins

&ty Formatting the trafficability features... (feature #5)
&r 154traf

/* Formatting the hydro and trans coverages for use later
/* This has to happen before 154hydro and 154trans!!!
&r 154hydtr

/* Formatting the arc features

```

```

&ty Formatting the roads... (feature #1)
&r 154trans

&ty Formatting the rivers... (feature #2)
&r 154hydro

/* Formatting the point features
/* Formatting the bridge features... (feature #7)
&r 154bridges

/* Formatting the polygon features
&ty Formatting the urban features... (feature #4)
&r 154urban

/* Creating the other necessary files: parms, map, line, rad, templates
&sys ..../amls/154others.perl
&sys cp xxline0.txt %.dirname%/.name%line0.txt
&sys cp xxrad0.txt %.dirname%/.name%rad0.txt
&sys cp xxline1.txt %.dirname%/.name%line1.txt
&sys cp xxrad1.txt %.dirname%/.name%rad1.txt
&sys cp xxline2.txt %.dirname%/.name%line2.txt
&sys cp xxrad2.txt %.dirname%/.name%rad2.txt
&sys cp xxlines.txt %.dirname%/.name%lines.txt
&sys cp xxrads.txt %.dirname%/.name%rads.txt
&sys rm xxrad*.txt
&sys rm xxline*.txt

&ty Creating an empty cbs 153 .trn file if necessary
&if ^ %153test% &then &r arccbs153 1
/* ****
&ty Cleaning up...
/* ****

&s total %total_t1% + %total_t2% + %total_t3% + %total_t4%
&s tilelist [open xtilelist.txt os -read]
&s junk [read %tilelist% rs]
&do loop3 = 1 &to %total%
  &s tile [read %tilelist% rs]
  &s a [locase [substr %tile% 1 1]]
  &s b [locase [substr %tile% 3 1]]
  &s c [locase [substr %tile% 5 2]]
  &s tile2 %a%/b%/c%
  &s cover %a%%b%%c%
  &do cov &list xpcov%cover% xrd%cover% xdn%cover% xpo%cover% xpppt%cover%
  /*      &if [exists %cov% -cover] &then kill %cov% all
  &end
&end
&s junk [close %tilelist%]

/* Deleting needed coverages

&do cov &list xtiles xtilesutm xclipcov xclipproj ~
xtemp xurbanx xurbanxg xtransx xtransxg xtransxgpt xtransxlp~
xpptemp xceanx xlakesx xlakesxg xdn xpo xpp xpppt
/*      &if [exists %cov% -cover] &then kill %cov% all
&end

```

### B.3 arccbs153.aml

```

&args dcw
&s junk [close -all]

/* arccbs153.aml
* -----
/* Program: arccbs153.aml
/* Purpose: to create a cbs 153 file (with empty hexes)
* -----
/* Called By: dcwcbs154.aml
/* Calls Made: none
* -----
/* Arguments: dcw = 1 if being called from dcwcbs154.aml
/* Globals: .hexsize - size of the hexes in meters
/* Locals:
* -----
/* Inputs: Arc coverages from user or DCW
/* Outputs: a CBS 153 terrain file (.trn file)
* -----
/* History:
/* This is an AML to convert ARC/INFO
/* coverages to CBS terrain files. Created by Erik Patton
/* and Tobi Steinberg of CUBIC TSI in 0495.
/* Need longlength.txt in the same workspace...
/* We are only creating lambert .trn files! None in UTM format.

/* Re-debugged and finished up by Tobi Steinberg in 0496.
/* Major changes:
/*   1. Changed all attribute assignments from selects to relates
/*   2. Building hexes in LCC now instead of UTM as is the standard
/*      for the type of .trn file we are building.
/*
/* 5 May 1997
/* Changed for use with cbs154 - taking out most of the guts
/* Might change to construct hex coverage with perl doing calcs
* -----
&messages &on
&echo &on

&if [null %dcw%] &then &do
  &if [exists arccbs153[date -tag].wat -file] &then
    &s delwat [delete arccbs153[date -tag].wat]
    &watch arccbs153[date -tag].wat
    &ty [date -VFULL]
  &end

/* =====
/* Deleteing needed coverage names and files and prompting for input.
/* =====

&do covs &list xhpoints xhpoints2 xtpoints ~
xtpoints2 xvegoutx xlakbigx xtempx xlakearcs ~
xhexx xfinriv xfinriv2 xlriver xmdriver xsmriver xurboutx xhexgeog ~
xtmpmerge xprvmerge rawelevgrid hexbnd hexbnddecsec rawelevgridc ~

```

```

elevgrid xidcov xelevcov xtmpmerge xpjinfo xutmcalcs xutmcalcs2 xvegx ~
elevgrids
  &if [exists %covs% -cover] &then kill %covs% all
  &else &if [exists %covs% -grid] &then kill %covs% all
  &end

&do files &list xrivout xrivout2 xtrout xvegout xlakout ~
xtdedcells.txt xtdedreq.txt xerrors.txt xerrors2.txt xroughness.txt ~
xroad{!1 2 3 4 5 6!}.txt xrivout{!1 2 3 4 5 6!} xcdin.txt xcdout.txt ~
xroad{!1 2 3 4 5 6!}b.txt xutm.txt xgeo.txt xgeo.prj xlcc.txt
  &if [exists %files% -file] &then
    &if [delete %files% -file] = 0 &then
      &ty Deleted file %files%
    &else &ty Problems deleting file %files%
  &end

&do info &list xroughness xriv{!1 2 3 4 5 6!} xrd{!1 2 3 4 5 6!} xlakes
xurban
  &if [exists %info%.info -info] &then
    &if [delete %info%.info -info] = 0 &then
      &ty Deleted info table %info%.info
    &else &ty Problems deleting info table %info%.info
  &end

&if ^ [null %dcw%] &then &do
  &s lakcov xlakesx
  &s hydcov xhydrox
  &s trcov xtransx
  &s urbcov xurbanx
  create xvegx xurbanx
  build xvegx
  &s vegcov xvegx
  &s size %.hexsize%
  &s hexcovname xxxxxhex
  &s name %.name%
  &do files &list %name%.trn x%name%.trn xx%name%.trn
    &if [exists %files% -file] &then &do
      &if [delete %files% -file] = 0 &then
        &ty Deleted file %files%
      &else &ty Problems deleting file %files%
    &end
  &end
  &s cm %.cm%
  &s spl %.spl%
  &s sp2 %.sp2%
  &end

/* cut out lots of stuff here from original arccbs.aml
&s xmin 999999999
&s xmax -999999999
&s ymin 999999999
&s ymax -999999999
build xhydrox line
build xtransx line
build xurbanx

```

```

build xvegx

&do covs &list xhydrox xtransx xurbanx xvegx
  &describe %covs%
    &if %dsc$xmin% lt %xmin% &then &s xmin %dsc$xmin%
    &if %dsc$xmax% gt %xmax% &then &s xmax %dsc$xmax%
    &if %dsc$ymin% lt %ymin% &then &s ymin %dsc$ymin%
    &if %dsc$ymax% gt %ymax% &then &s ymax %dsc$ymax%
  &end

/* Prompting for the hex coverage name.
&label badqhex
&if [exists xxxxxhex -cover] &then &goto dcwcontinue
&ty If you have changed your playbox extents, you will need to
&ty create a new hex coverage.
&s qhex = [response 'Do you have a hexagon coverage? (y/n)']
&select %qhex%
  &when y
    &do
      &s hexcovname [response 'Enter the name of ~
                      your hexagon coverage [%name%hex]' %name%hex]
      &if [exists %hexcovname% -clean] &then &do
        copy %hexcovname% xhexx
        &if ^ [exists xhexx -clean] &then build xhexx
      &end
      &else
        &do
          &ty 'Coverage' %hexcovname% 'does not exist or does not
have good topology.'
          &goto badqhex
        &end
      &describe xhexx
      &s size_km [calc %size% / 1000]

    &end
  &when n
    &do
      /*-----
      /* Creating the hex coverage
      -----*/
      &ty beginning of hex generation
      &ty [date -VFULL]

      /* Query user for initial parameters
      &label badrespl
      &s respl [response 'Would you like to save your hex coverage
[y]' y]
        &if %respl% ne y and %respl% ne n &then &do
          &ty Please type a y or n.
          &goto badrespl
        &end

        &if %respl% = y &then &do
          &label badhexcovname
          &s hexcovname = [response 'What would you like to call your
hex coverage [hex1 hex1]

```

```

/*
/* E = Location marked by variables xrighthor and yrighthor
/*
/* The line segment from B to D is defined as a "left" line and the
segment
/* from C to E is defined as a "right" line. Here left and right
pertains
/* only to individual hexes.
/*
/* The distance from D to F is one quarter of the distance accross the
hex
/* and is defined in variable qsize.
/*
/* The distance from B to F is the square root of 3 * the distance from
D to
/* F (qsize).
/*
/* A left hex is one in an odd numbered column and a right hex is one in
an
/* even numbered column.
/*
/* Hexes are numbered according to (column,row) as indicated above.

&s size_km [calc %size% / 1000]
&s size [calc %size% * 1.1547005]
&s qsize = [calc %size% / 4]
&s rtsize = [calc %qsize% * [sqrt 3]]

/* Determine the number of columns to create
&s xdelta = [calc %xmax% - %xinit%]
&s 2colwidth = [calc %qsize% * 6]
&s no2cols = [calc %xdelta% / %2colwidth%]
&s columns = [round [calc %no2cols% * 2]]
&if [mod %columns% 2] ne 0 &then
    &s columns = [calc %columns% + 1]

/* Calculate the number of rows to create
&s ydelta = [calc %ymax% - %yinit%]
&s hexheight = [calc %rtsize% * 2]
&s rows = [round [calc %ydelta% / %hexheight%]]

/* Turn initial values into working variables
/* Set variables for left horizontal lines
&s xlefthor = [calc %xinit% + %qsize%]
&s ylefthor = %yinit%

/* Set variables for right horizontal lines
&s xrighthor = [calc %xinit% + [calc %qsize% * 4]]
&s yrighthor = [calc %yinit% + %rtsize%]

/* Set variables for left vertical lines
&s xleftvert = [calc %xinit% + %qsize%]
&s yleftvert = %yinit%

/* Set variables for right vertical lines
&s xrightvert = [calc %xinit% + [calc %qsize% * 3]]
&s yrighvert = %yinit%

```

```

/* Use GENERATE to create the coverages
generate %hexcovname%
lines

&echo &off

/* Draw bulk of the hexes -----
&do norows = 1 &to %rows%
&ty %norows%; &ty %rows%

/* Draw the left horizontal lines for the number of columns
&do nocols = 1 &to [calc %columns% / 2]

/* Add more lines
%nocols%
%ylefthor%,%ylefthor%
[calc %ylefthor% + [calc %qsize% * 2]],%ylefthor%

/* Move the starting point.
&s xlefthor = [calc %ylefthor% + [calc %qsize% * 6]]
&s ylefthor = %ylefthor%
&s ytop = [calc %ylefthor% + [calc %rtsize% * 2]]
end

&end

/* Draw the right horizontal lines for the number of columns.
&do nocols = 1 &to [calc %columns% / 2]

/* Add more lines
%nocols%
%xrighthor%,%yrighthor%
[calc %xrighthor% + [calc %qsize% * 2]],%yrighthor%

/* Move the starting point.
&s xrighthor = [calc %xrighthor% + [calc %qsize% * 6]]
&s yrighthor = %yrighthor%
end

&end

/* Draw the left vertical lines for the number of columns.
&do nocols = 1 &to [calc %columns% / 2]

/*lines
%nocols%
%yleftvert%,%yleftvert%
[calc %yleftvert% - %qsize%],[calc %yleftvert% + %rtsize%]
%yleftvert%,[calc %yleftvert% + [calc %rtsize% * 2]]

/* Move the starting point
&s xleftvert = [calc %yleftvert% + [calc %qsize% * 6]]
&s yleftvert = %yleftvert%
end

```

```

&end

/* Draw the right vertical lines for the number of columns.
&do nocols = 1 &to [calc %columns% / 2]

/*lines
%nocols%
%rightvert%,%rightvert%
[calc %xrightvert% + %qsize%],[calc %yrightvert% +
%rtsize%]

%rightvert%,[calc %yrightvert% + [calc %rtsize% * 2]]

/* Move the starting point
&s xrightvert = [calc %xrightvert% + [calc %qsize% * 6]]
&s yrightvert = %yrightvert%
end

&end

/* Change the starting points for the row. Move UP one.
&s xlefthor = [calc %xinit% + %qsize%]
&s ylefthor = [calc %ylefthor% + [calc %rtsize% * 2]]
&s xrighthor = [calc %xinit% + [calc %qsize% * 4]]
&s yrighthor = [calc %yrighthor% + [calc %rtsize% * 2]]
&s xleftvert = [calc %xinit% + %qsize%]
&s yleftvert = [calc %ylefthor% + [calc %rtsize% * 2]]
&s xrightvert = [calc %xinit% + [calc %qsize% * 3]]
&s yrightvert = [calc %yrightvert% + [calc %rtsize% * 2]]

&end

/* Add two lines at the end of each row -----
/* Set initial variables

&s e_cols = [calc %columns% / 2]
&s e_dist = [calc %qsize% * 6]
&s xextra = [calc %e_cols% * %e_dist% + %xinit%]
&s yextra = [calc %yinit% + %rtsize%]

%nocols%
&do I = 1 &to [calc %norows% - 1]

%extra%,%yextra%
[calc %xextra% + %qsize%],[calc %yextra% + %rtsize%]
%extra%,[calc %yextra% + [calc %rtsize% * 2]]

/* Move the starting point
&s xextra = %xextra%
&s yextra = [calc %yextra% + [calc %rtsize% * 2]]

&end
end

/* Finish off the top -----
/* Do the left top first
&do nocols = 1 &to [calc %columns% / 2]

```

```

/* Add the tops of the columns
%nocols%
%xlefthor%,%ylefthor%
[calc %xlefthor% + [calc %qsize% * 2]],%ylefthor%
end

/* Move the starting point
&s xlefthor = [calc %xlefthor% + [calc %qsize% * 6]]
&s ylefthor = %ylefthor%

&end

/* Now do the right top
&do nocols = 1 &to [calc %columns% / 2]

/* Add the tops of the columns
%nocols%
%righthor%,%yrighthor%
[calc %xrighthor% + [calc %qsize% * 2]],%ylefthor% +
%rtsize%]
end

/* Move the starting point
&s xrighthor = [calc %xrighthor% + [calc %qsize% * 6]]
&s yrighthor = %yrighthor%

&end

/* Do the left vertical line
&do nocols = 1 &to [calc %columns% / 2]

%nocols%
[calc %xleftvert% + [calc %qsize% * 2]],%yleftvert%
[calc %xleftvert% + [calc %qsize% * 3]],%yleftvert% +
%rtsize%]
end

/* Move the starting point
&s xleftvert = [calc %xleftvert% + [calc %qsize% * 6]]
&s yleftvert = %yleftvert%

&end

/* Do the right vertical line
&do nocols = 1 &to [calc [calc %columns% / 2] - 1]

%nocols%
[calc %xrightvert% + [calc %qsize% * 4]],%yrightvert%
[calc %xrightvert% + [calc %qsize% * 3]],%yrightvert% +
%rtsize%]
end

/* Move the starting point
&s xrightvert = [calc %xrightvert% + [calc %qsize% * 6]]
&s yrightvert = %yrightvert%

&end

```

```

end

/* Add label points -----
points

/* LEFT HEX POINTS
/* Turn initial values into working variables
&s xpt = [calc %xinit% + [calc %qsize% * 2]]
&s ypt = [calc %yinit% + %rtsize%]
&s lefthexinc = 1

/* Do the left hex points first
&do xcount = 1 &to [calc %columns% / 2]

&do ycount = 1 &to %rows%

/* The id values are column * 1000 + the row
[calc [calc %lefthexinc% * 1000] + %ycount%],%xpt%,%ypt%

/* Move the starting point
&s xpt = %xpt%
&s ypt = [calc %ypt% + [calc %rtsize% * 2]]

&end

/* Move the starting point
&s xpt = [calc %xpt% + [calc %qsize% * 6]]
&s lefthexinc = %lefthexinc% + 2
&s ypt = [calc %yinit% + %rtsize%]

&end

/* RIGHT HEX POINTS
/* Turn initial values into working variables
&s xpt = [calc %xinit% + [calc %qsize% * 5]]
&s ypt = [calc %yinit% + [calc %rtsize% * 2]]
&s righthexinc = 2

/* Now do the right hex points
&do xcount = 1 &to [calc %columns% / 2]

&do ycount = 1 &to %rows%

/* The id values are column * 1000 + the row
[calc [calc %righthexinc% * 1000] + %ycount%],%xpt%,%ypt%

/* Move the starting point
&s xpt = %xpt%
&s ypt = [calc %ypt% + [calc %rtsize% * 2]]

&end

/* Move the starting point and increment the count by two
&s xpt = [calc %xpt% + [calc %qsize% * 6]]
&s righthexinc = %righthexinc% + 2
&s ypt = [calc %yinit% + [calc %rtsize% * 2]]

```

```

&end

/* End GENERATE input and quit
end
quit

&echo &on
&ty end of hex generation
&ty [date -VFULL]

/* Defining the projection and coordinate system
&ty Defining the projection and coordinate system ...
projectdefine cover %hexcovname%
projection lambert
units meters
spheroid %spheroid%
parameters
%sp1% 0 0
%sp2% 0 0
%cm% 0 0
0 0 0
0
0

/* Clean the coverage with a fuzzy of 3
&ty Cleaning the coverage ...
clean %hexcovname% %hexcovname% 0 3 poly

/* Redefine the poly ids to get row and column ids and add other
attributes
&ty Populating the row and column values ...
additem %hexcovname%.pat %hexcovname%.pat hexnum 6 6 i
tables
sel %hexcovname%.pat
calc hexnum = %hexcovname%-id
redefine
17
hex_x
3
3
i
20
hex_y
3
3
i
~
quit

rename %hexcovname% xhexx

/* =====
/* Adding necessary items to the hex coverage
/* =====

additem xhexx.pat xhexx.pat terrain 2 2 I
additem xhexx.pat xhexx.pat urban 2 2 I

```

```

additem xhexx.pat xhexx.pat veg 2 2 I
additem xhexx.pat xhexx.pat elev 4 4 I
additem xhexx.pat xhexx.pat rough 4 4 I
additem xhexx.pat xhexx.pat dissolve 1 1 I

&do item &list road1 road2 road3 road4 road5 road6 ~
river1 river2 river3 river4 river5 river6 ~
bridge1 bridge2 bridge3 bridge4 bridge5 bridge6
    additem xhexx.pat xhexx.pat %item% 2 2 I
&end

additem xhexx.pat xhexx.pat pound 2 2 C
copy xhexx %hexcovname%

&if %dcw% ne 1 &then &goto dcwcontinue

&end
&otherwise
&do
    &ty 'Please answer y or n.'
    &goto badqhex
&end
&end

&label dcwcontinue
&label defbegin
&if %dcw% = 1 &then
    &if [exists xxxxhex -cover] &then &do
        copy xxxxhex xhexx
        build xhexx poly
    &end

/* =====
/* Snapping the rivers to hex sides.
/* =====

/* (deleted)

/* =====
/* Writing the river attributes to the hex file.
/* =====

/* (deleted)

/* =====
/* Creating the roads hex numbering sequence file.
/* =====

/* (deleted)

/* =====
/* Writing the road attributes to the hex file.
/* =====

/* (deleted)

```

```

/* =====
/* Writing the urban attributes to the hex file.
/* =====

/* (deleted)

/* =====
/* Writing the veg attributes to the hex file.
/* =====

/* (deleted)

/* =====
/* Calculating which hex sides have bridges.
/* =====

/* (deleted)

/* =====
/* Writing the roughness and elevation information to the hex file.
/* =====

/* (deleted)

/* =====
/* Writing the terrain information to the hex file.
/* =====

/* (deleted)

/* -----
/* =====
/* The beginning of the end...
/* -----
/* ----

&s junk [close -all]

/* =====
/* Unloading the hex data to create the .TRN file
/* =====

&ty Creating the first text file containing geographic and ~
projection header information...

tables
    &s spaces40 ' '

```

```

/* Moving the calculation of the number of rows and columns
/* here because I can't access the sort command from within
/* arcedit; also so I don't have to restart tables again.

sel xhexx.pat
sort hexnum (D)
&s columns [show record 1 item hex_x]
&s rows [show record 1 item hex_y]
sort xhexx#

```

```

quit

/* I'm going to do this by creating a coverage and adding
/* the projection information in an item in arcedit. This will
/* make formatting easier by being able to use the unload
/* command.

create xpjinfo xhexx
build xpjinfo poly
additem xpjinfo.pat xpjinfo.pat initspc 12 12 c
additem xpjinfo.pat xpjinfo.pat projinfo 15 15 n 7
additem xpjinfo.pat xpjinfo.pat finspc 40 40 c
additem xpjinfo.pat xpjinfo.pat junk 5 5 n

/* The first twelve calculations necessitate the create of yet another
/* new coverage.

create xutmcalcs

/* Determining the boundaries of the central UTM zone of the data
/* and also the central parallel of the data.

&describe xhexx
/* going through this dumb process to pull out the first number... duh
&s cm 0
&do n &list %prj$cm%
  &if %cm% = 0 &then
    &if %n% ne 0 &then &s cm %n%
  &end

&s sp1 0
&s sp2 0
&do n &list %prj$sp1%
  &if %sp1% = 0 &then
    &if %n% ne 0 &then &s sp1 %n%
  &end

&do n &list %prj$sp2%
  &if %sp2% = 0 &then
    &if %n% ne 0 &then &s sp2 %n%
  &end

&s centpar [round [calc [calc %sp1% + %sp2%] / 2]]
/* 1 corresponds to the northern hemisphere; 2 to the southern
&if %centpar% gt 0 &then &s hemi 1
&else &s hemi 2

&s zone [calc %cm% / 6]
&s zone [calc [calc %cm% + 186] / 6]

&if %zone% gt 30 &then &s rzonedge [calc [calc %zone% - 30] * 6]
&else &s rzonedge [calc [calc [calc %zone% - 31] * 6] + 6]
&s lzonedge [calc %rzonedge% - 6]

tables
  sel xutmcalcs.tic
  add

  &s xtc [calc %lzonedge% + .25]
  &s ytc [calc %centpar% - .25]
1
%xtc%
%ytc%
  &s xtc [calc %rzonedge% + .25]
2
%xtc%
%ytc%
  &s xtc [calc %lzonedge% + .25]
3
%xtc%
%centpar%
  &s xtc [calc %rzonedge% + .25]
4
%xtc%
%centpar%
  &s xtc [calc %lzonedge% - .25]
5
%xtc%
%centpar%
  &s xtc [calc %rzonedge% - .25]
6
%xtc%
%centpar%
  ~
quit

project cover xutmcalcs xutmcalcs2
input
projection geographic
units dd
spheroid int1967
parameters
output
projection utm
units meters
spheroid int1967
parameters
%cm% 0 0
%centpar% 0 0
end

tables
  sel xutmcalcs2.tic
  &do ticnum = 1 &to 6
    &s utm%ticnum%x [show record %ticnum% item xtic]
    &s utm%ticnum%y [show record %ticnum% item ytic]
  &end
quit

/* Going into arcedit in order to calculate and set the values
/* of each piece of header file information.

ae
  ec xhexx
  ef poly

```

```

sel $recno lt 38
put xpjinfo
y
ec xpjinfo
build
ef poly
sel all
calc junk = $recno - 37

/* (still need to alter this section for the southern hemisphere)

/* Point number one - lower right
/* 1 Left east-west UTM coordinate
sel junk = 2
calc projinfo = %utm1x%

/* 2 Left north-south UTM coordinate
sel junk = 3
calc projinfo = %utm1y%

/* 3 Right east-west UTM coordinate
sel junk = 4
calc projinfo = %utm2x%

/* 4 Right north-south UTM coordinate
sel junk = 5
calc projinfo = %utm2y%

/* Point number two - upper right
/* 5 Left east-west UTM coordinate
sel junk = 6
calc projinfo = %utm3x%

/* 6 Left north-south UTM coordinate
sel junk = 7
calc projinfo = %utm3y%

/* 7 Right east-west UTM coordinate
sel junk = 8
calc projinfo = %utm4x%

/* 8 Right north-south UTM coordinate
sel junk = 9
calc projinfo = %utm4y%

/* Point number three - upper left
/* 9 Left east-west UTM coordinate
sel junk = 10
calc projinfo = %utm5x%

/* 10 Left north-south UTM coordinate
sel junk = 11
calc projinfo = %utm5y%

/* 11 Right east-west UTM coordinate
sel junk = 12
calc projinfo = %utm6x%

```

---

```

/* 12 Right north-south UTM coordinate
sel junk = 13
calc projinfo = %utm6y%

/* Lambert Conformal Conic projection parameters
/* Southern standard parallel
sel junk = 14
calc projinfo = %sp1%

/* 14 Northern standard parallel
sel junk = 15
calc projinfo = %sp2%

/* 15 Central meridian
sel junk = 16
calc projinfo = %cm%

/* Moving the calculation for item 26 up to here because its value
/* is needed.
/* 26 Pixel distance from northern to southern playbox boundaries
&s pix_unit [calc [calc [%size_km% * 1000] / 200] / 2]
&s ypixdis [calc 3 + %pix_unit% + [calc 2 * %pix_unit% * %rows%]]

/* create another coordinate text file to find out the exact upper
/* and lower limits of the hex coverage in geographic coordinates
/* and the eastern and western limits
&if [exists xgeo.prj -file] &then &s junk [delete xgeo.prj -file]
&if [exists xgeo.txt -file] &then &s junk [delete xgeo.txt -file]
&describe xhexx
&s space ' '
&s cdin [open xlcc.txt os -write]
&s junk [write %cdin% 0%space%%dsc$ymax%]
&s junk [write %cdin% 0%space%%dsc$ymin%]
&s junk [write %cdin% %dsc$ymax%%space%%dsc$xmin%]
&s junk [write %cdin% %dsc$ymax%%space%%dsc$xmax%]
&s junk [close %cdin%]

save
quit

project file xlcc.txt xgeo.txt
  input
    projection lambert
    units meters
    spheroid %prj$spheroid%
  parameters
    %prj$sp1%
    %prj$sp2%
    %prj$cm%
    0 0 0
    0
    0
  output
    projection geographic
    units dd
    spheroid %prj$spheroid%

```

```

parameters
end

&s cdin [open xgeo.txt os -read]
&s yhi [trim [after [trim [read %cdin% rs] -BOTH %space%] %space%] -BOTH
%space%]
&s ylo [trim [after [trim [read %cdin% rs] -BOTH %space%] %space%] -BOTH
%space%]
&s xhi [trim [before [trim [read %cdin% rs] -BOTH %space%] %space%] -BOTH
%space%]
&s xlo [trim [before [trim [read %cdin% rs] -BOTH %space%] %space%] -BOTH
%space%]
&s junk [close %cdin%]

/* Compute 'Lambert Numbers' to be used in 16, 18, and 20.

/* Now calculating the polar radius values of the upper and
/* lower playbox limits along the central meridian. For the
/* northern hemisphere, temp3 is above temp1. The numbers are
/* both negative because we are using the center of the cone -
/* the pole - as our origin. The numbers are essentially unitless
/* because we are using SCALE * EARTH RADIUS = 1

/* for more information on the lambert equations, see any good
projections
/* reference. I am using the spheroidal model.

/* First, calculating the constant of the cone, n
/* a. set phi values - standard parallel locations
&if %sp2% gt %sp1% &then &do
    &s phi_1 [angrad %sp1%]
    &s phi_2 [angrad %sp2%]
&end
&else &do
    &s phi_1 [angrad %sp2%]
    &s phi_2 [angrad %sp1%]
&end

&s pi 3.141592658

/* b. calculate the upper and lower sections of the cone constant
equation
&s a [calc [cos %phi_1%] / [cos %phi_2%]]
&s b [calc [tan [calc [calc %pi% / 4] + [calc %phi_2% / 2]]] / ~
    [tan [calc [calc %pi% / 4] + [calc %phi_1% / 2]]]]
&s lna [calc ln %a%]
&s lnb [calc ln %b%]

/* ... and the constant itself
&s n [calc %lna% / %lnb%]

/* c. now calculating a sub item needed in the equation, that I'll
/* call G. I'm going to go ahead and calculate it for phi_lo, phi_hi,
/* and phi_1. It is independent of phi_2.

&s phi_hi [angrad %yhi%]
&s phi_lo [angrad %ylo%]

&s g1 [tan [calc [calc %pi% / 4] + [calc %phi_hi% / 2]]]
&s g2 [tan [calc [calc %pi% / 4] + [calc %phi_lo% / 2]]]
&s g3 [tan [calc [calc %pi% / 4] + [calc %phi_1% / 2]]]

&s g_hi [calc %g1% ** %n%]
&s g_lo [calc %g2% ** %n%]
&s g_1 [calc %g3% ** %n%]

&s temp3 [calc [cos %phi_1%] * %g_1% / [calc %g_hi% * %n%]]
&s temp1 [calc [cos %phi_1%] * %g_1% / [calc %g_lo% * %n%]]

/* d. now I have the building blocks to actually calculate the lambert
/* numbers at the top and the bottom of the playbox along the
/* central meridian

ae
ec xpjinfo
ef poly

/* 16 Lambert to pixel conversion constant in the y-direction
&if %hemi% = 1 &then
    &s out1 [calc %ypixdis% / [calc %temp1% - %temp3%]]
&else &s out1 [calc %ypixdis% / [calc %temp3% - %temp1%]]
sel junk = 17
calc projinfo = %out1%

/* 17 Orthogonal projection (always 0)
sel junk = 18
calc projinfo = 0

/* 18 pixel distance from cone apex to northern playbox boundary
/* using the value of out3, as described in the original TDPS code
&s out3 [calc %out1% * %temp3%]
&if %hemi% = 2 &then &s out3 [calc %out1% * %temp1%]
sel junk = 19
calc projinfo = %out3%

/* 19 Orthogonal projection (always 0)
sel junk = 20
calc projinfo = 0

/* 20 Lambert to pixel conversion constant (x-direction) (same as y)
&if %hemi% = 2 &then &s out1 [calc %out1% * -1]
sel junk = 21
calc projinfo = %out1%

/* 21 Pixel distance from central meridian to western playbox boundary
/* Determining the distance of a degree at this latitude in meters
&s lentxt [open longlength.txt os -read]
&s ymiddle [abs [truncate %ylo%]]
&if %ymiddle% ne 0 &then &do t = 0 &to [calc %ymiddle% - 1]
    &s foo [read %lentxt% rs]
    &s foo [read %lentxt% rs]
&end
&s foo [read %lentxt% rs]
&s deglen [read %lentxt% rs]

```

```

&s junk [close %lentxt%]
&s xdiff [calc [calc [calc %xlo% - %cm%] * %deglen%] / 200]
sel junk = 22
calc projinfo = %xdiff%

/* 22 Half the height of a hex in pixels
sel junk = 23
calc projinfo = %pix_unit%

/* 23 Distance between adjacent hex centers in pixels
&s pix_fact [calc [calc [calc 2 * %pix_unit%] / [sqrt %size_km%]] * 1.5]
sel junk = 24
calc projinfo = %pix_fact%

/* 24 X-offset from hex (1,1) center to western playbox boundary
sel junk = 25
calc projinfo = 0

/* 25 Y-offset from hex (1,1) center to southern playbox boundary
sel junk = 26
calc projinfo = 0

/* 26 (Moved calculation above; number needed in a previous computation)
sel junk = 27
calc projinfo = %ypixdis%

/* 27 Total number of hexes in the east-west direction
sel junk = 28
calc projinfo = %columns%

/* 28 Total number of hexes in the north-south direction
sel junk = 29
calc projinfo = %rows%

/* 29 Hex size (length from side to side not corner to corner)
sel junk = 30
calc projinfo = %size_km%

/* 30 East-west starting hex number
sel junk = 31
calc projinfo = 1

/* 31 North-south starting hex number
sel junk = 32
calc projinfo = 1

/* 32 East-west ending hex number
sel junk = 33
calc projinfo = %columns%

/* 33 North-south ending hex number
sel junk = 34
calc projinfo = %rows%

/* 34 "Not used for anything and I don't know what it is..."
sel junk = 35
calc projinfo = 0

```

```

/* 35 "Not used for anything and I don't know what it is..."
sel junk = 36
calc projinfo = 0

/* 36 Pixel resolution in meters
sel junk = 37
calc projinfo = 200
save
quit

&ty Creating the second text file containing the hex information...
additem xhexx.pat xhexx.pat hex_x2 4 4 i
additem xhexx.pat xhexx.pat hex_y2 4 4 i
tables
sel xhexx.pat
calc hex_x2 = hex_x
calc hex_y2 = hex_y
sort hexnum
move ' #' to pound
resel $recono ne 1
unload xx%name%.trn terrain urban veg elev rough ~
road1 road2 road3 road4 road5 road6 ~
river1 river2 river3 river4 river5 river6 bridge1 ~
bridge2 bridge3 bridge4 bridge5 bridge6 pound hex_x2 hex_y2 columnar
format
sel xhexx.pat
sort xhexx#
sel xpjinfo.pat
move ' ' to finspc
move ' ' to initspc
resel $recono ne 1
unload x%name%.trn initspc projinfo finspc columnar format
quit

&s junk [delete format -file]

/* Combining the two text files to create a single output file.
&sys cat x%name%.trn xx%name%.trn > %name%.trn
&sys cp %name%.trn %.dirname%/%name%.trn
&if [exists %.dirname%/%name%.trn -file] &then &s junk [delete %name%.trn
-file]
&ty Cleaning up ...

&do file &list x%name%.trn xx%name%.trn xrivout xrivout2 xlakout xtrout ~
xtdedrej.txt xtdedrej2.txt xtdedcells.txt xerrors.txt xroughness.txt ~
xtrout xrivout xcdin.txt xcdout.txt xutm.txt xgeo.txt xlcc.txt ~
xroad1.txt xroad2.txt xroad3.txt xroad4.txt xroad5.txt xroad6.txt ~
xroad1b.txt xroad2b.txt xroad3b.txt xroad4b.txt xroad5b.txt xroad6b.txt ~
xriver1.txt xriver2.txt xriver3.txt xriver4.txt xriver5.txt xriver6.txt
&if [exists %file% -file] &then &s junk [delete %file% -file]
&end

&do covs &list xhydrox xhpoints xhpoints2 xtransx xtpoints ~
xtpoints2 xurbanx xvegx xvegoutx xlakesx xlakbigx ~
xfinriv xfinriv2 xlriver xmdriver xsriver xurboutx xhexgeog ~
xtmpmerge xprvmerge xutmcalcs xutmcalcs2 xhexx xpjinfo xidcov ~

```

```

xelevcov xttmpmerge xlakearcs
  &if [exists %covs% -cover] &then kill %covs% all
&end

&do info &list xroughness xlakes
  &if [exists %info%.info -info] &then
    &ty [delete %info%.info -info]
  &end

  &if ^ [null %dcw%] &then &return
  &ty Done.
  &ty [date -VFULL]
  &watch &off
  &s delwat [delete arccbs153[date -tag].wat]
  &return

=====
longlength.txt - needs to be in the current workspace
=====

0
111321
1
111304
2
111253
3
111169
4
111051
5
110900
6
110715
7
110497
8
110245
9
109959
10
109641
11
109289
12
108904
13
108486
14
108036
15
107553
16
107036
17
106487
18

```

```

105906
19
105294
20
104649
21
103972
22
103264
23
102524
24
101754
25
100952
26
100119
27
99257
28
98364
29
97441
30
96488
31
95506
32
94495
33
93455
34
92387
35
91290
36
90166
37
89014
38
87835
39
86629
40
85396
41
84137
42
82853
43
81543
44
80208
45
78849
46
77466

```

47	28903
76058	76
48	27017
74628	77
49	25123
73174	78
50	23220
71698	79
51	21311
70200	80
52	19394
68680	81
53	17472
67140	82
54	15545
65578	83
55	13612
63996	84
56	11675
62395	85
57	9735
60774	86
58	7792
59135	87
59	5846
57478	88
60	3898
55802	89
61	1949
54110	
62	
52400	
63	
50675	
64	
48934	
65	
47177	
66	
45407	
67	
43622	
68	
41823	
69	
40012	
70	
38188	
71	
36353	
72	
34506	
73	
32648	
74	
30781	
75	

## B.4 154hydtr.aml

```
/* -----
/* Program: 154hydtr.aml
/* Purpose: to make some adjustments to the hydro and trans
/* files that will be needed later by the bridges program
/* -----
/* Called By: dcwcbs154.aml
/* -----
/* Arguments: none
/* Globals: .name, .dirname
/* -----
/* Inputs: trans and hydro files
/* Outputs: adjusted versions of the same
/* -----
/* History: 18 August 1997 - Tobi Sellekaerts
/* steinbet@email.grafenwoehr.army.mil
/* Terrain Simulations - 7th ATC - Grafenwoehr, Germany
/* -----
&severity &error &routine bail
&echo &on
display 0
&ty [date -VFULL]

/* Checking on the existence of coverage names needed by the program
&do cov &list xhydroxg xtransxg xxtemp xxtemp2
  &if [exists %cov% -cover] &then kill %cov% all
&end

/* Unsplits the features of each coverage based on rdlntype for the
/* roads and
ae
  ec xtransx
  ef arc
  sel all
  calc xtransx-id = rdlntype
  unsplit
  save
  ec xhydrox
  ef arc
  sel dnlstat gt 0
  calc xhydrox-id = dnlstat
  unsplit
  save
quit

build xtransx line
build xhydrox line

tables
  sel xtransx.aat
  calc xtransx-id = xtransx# - 1
  sel xhydrox.aat
  calc xhydrox-id = xhydrox# - 1
quit

/* Add the features from each coverage into the other so that there
```

```
/* are nodes at every intersection.

/* First, change tolerances
&do cov &list xhydrox xtransx
  tolerance %cov% nodesnap 1
  tolerance %cov% dangle 1
  tolerance %cov% weed 1
&end

/* make copies for use in the next section
copy xtransx xxtemp
build xxtemp line
copy xhydrox xxtemp2
build xxtemp2 line

/* Second, place items into the other coverages
ae
  ec xtransx /* working on transportation first
  ef arc
  sel all
  delete
  ec xhydrox
  ef arc
  sel all
  put xtransx
  y
  ec xxtemp
  ef arc
  sel all
  put xtransx
  y
  save all yes

  ec xhydrox /* working on hydro second
  ef arc
  sel all
  delete
  ec xxtemp
  ef arc
  sel all
  put xhydrox
  y
  ec xxtemp2
  ef arc
  sel all
  put xhydrox
  y
  save all yes
quit

kill xxtemp all
kill xxtemp2 all

/* Third, build each coverage so the necessary nodes are added
clean xhydrox
clean xtransx
```

```

/* Fourth, delete the opposing features so hydro doesn't actually
/* end up in the trans file, and vice versa.
ae
  ec xtransx
  ef arc
  sel rdlntype = 0
  delete
  save
  ec xhydrox
  ef arc
  sel dnlntype = 0
  delete
  save
quit

build xhydrox line
build xtransx line

/* Project both coverages into geographic for later use
&do cov &list xhydrox xtransx
  project cover %cov% %cov%g
    output
    projection geographic
    units dd
    spheroid wgs84
    datum wgs84
    parameters
  end
&end

build xhydroxg line
build xtransxg line

&echo &off
display 9999
&ty [date -VFULL]
&return
*****
/* bail routine
*****
&routine bail
&severity &error &ignore
&severity &warning &ignore
&ttype An error has occurred in 154hydtr.aml
&ttype Bailing out of 154hydtr.aml
&return; &return &error

```

## B.5 Transportation

### B.5.1 154trans.aml

```

/* 154trans.aml
/*
-----+
/*      Program: 154trans.aml
/*      Purpose: To format the arc features (roads) into
/*      files which can be used by CBS 154, and copy these files into
/*      the final output directory.
/* -----
/*      Called By: dcwcbs154.aml
/*      Calls Made: 154trans.perl
/* -----
/*      Arguments: none
/*      Globals: .dirname - the name of the output directory
/*                  .name - trunk name for the output files
/*      Locals:
/* -----
/*      Inputs: none
/*      Outputs: correctly formatted arc files for cbs 154
/* -----
/*      History: 6 May 1997 - Tobi Steinberg
/*                  steinbet@email.grafenwoehr.army.mil
/*                  http://199.56.131.202/~tobi/
/*      Terrain Simulations - 7th ATC - Grafenwoehr, Germany
/* -----
&severity &error &routine bail
&echo &on
display 0
&ty [date -VFULL]

&s incov xtransx

/* Checking on the existence of coverage names needed by the program
&do cov &list %incov%gpt %incov%pt %incov%lpt
  &if [exists %cov% -cover] &then kill %cov% all
&end

/* Deal with the coverage with all three component sizes together; split
/* later in the perl script

/* Addint items for x and y coordinates in both geographic and meters
/* Getting the meters values from Lambert since I already have a set of
/* coverages in that projection and can easily determine the parameters
arcpoint %incov%g %incov%gpt line %incov%g#
addxy %incov%gpt point
tables
  sel %incov%gpt.pat
  alter x-coord
    geox
    ~
    ~
    ~
    ~
  alter y-coord
    geoy

```

```

~
~
~
~

quit

/* Projecting the point coverage into lambertcc
&describe %incov%
project cover %incov%gpt %incov%lpt
    output
        projection lambert
        units meters
        spheroid wgs84
        datum wgs84
        parameters
            %prj$sp1%
            %prj$sp2%
            %prj$cm%
        ~
        ~
        ~
    end

countvertices %incov%g line
&sys rm xxtransin?.txt
&sys rm xxtrnode?.txt
addxy %incov%lpt point
tables
    sel %incov%lpt.pat
    unload xxtransin1.txt %incov%g# geox geoy x-coord y-coord columnar
junk.txt
    sel %incov%g.aat
    unload xxtransin2.txt %incov%g# vertices rdlntype length columnar
junk.txt
    &sys rm junk.txt
    sort fnode#
    unload xxtrnode1.txt %incov%g# fnode# columnar junk.txt
    sort tnode#
    unload xxtrnode2.txt %incov%g# tnode# columnar junk.txt
quit

/* Calling the perl script to format the file
&sys /data2/cbs/154/amls/154trans.perl

&sys rm xxtransin?.txt
&sys rm xxtrnode?.txt
&sys rm xxtrandout6.txt
&sys rm xxtransout?b.txt

display 9999
&ty [date -VFULL]
&return
*****
/* bail routine
*****
&routine bail
&severity &error &ignore

```

## B.5.2 154trans.perl

```

#! /usr/local/bin/perl

# -----
#     Program: 154trans.perl
#     Purpose: to format a text file output by 154trans.aml into the
#               correct format for cbs 154. Used for roads.
# -----
#     Called By: 154trans.aml
#     Calls Made: none
# -----
#     Arguments: none
#     Globals: none
#     Locals:
# -----
#     Inputs: text file created by 154trans.aml - point in
xxtransin1.txt
#           lines in xxtransin2.txt
#     Outputs: correctly formatted text file for cbs 154
# -----
#     History: 6 May 1997 - Tobi Steinberg
#               steinbet@email.grafenwoehr.army.mil
#               http://199.56.131.202/~tobi/
#               Terrain Simulations - 7th ATC - Grafenwoehr, Germany
# -----


# ######
# This first half of the programs creates the output files without the
# necessary 'merge' comments in them. It will also create a list of the
# number, start, and end point of each road. This list will be sorted
then
# used to add the necessary merge comments to the output files.
# ######



open(MERGE, ">xxtransout6.txt");

# creating a statistics file to be used by 154bridges2.perl later
open(RDSTAT, ">xxrstat.txt");

# Setting counts of files for roads of three different types
$tpct = 100001;
print RDSTAT "$tpct\n";

# Repeat this entire section three times. Once for line 1, then for 2,
then for 3
# open various files for reading and writing
open(PTIN,"xxtransin1.txt");      # point information in
open(LNIN,"xxtransin2.txt");      # line information in
open(OUT1, ">xxtransout1.txt");    # type 1 roads output file
$blah = 101;
# read information in, format it, and output to the appropriate files
while($inline = <LNIN>) {          # cycle as long as roads remain
    $verts = substr($inline, 5, 6);   # number of vertices
    $rdtype = substr($inline, 22, 1); # type of road
    $length = substr($inline, 23, 12); # overall length of the road
}

```

```

$verts *= 1;
$rdtype *= 1;
$length /= 1000;
$blah *= 1;

# print out the header line information
if($rdtype == 3) {
    print OUT1 " $blah $verts";
    printf OUT1 "%2d%7d%2d%2d%2d%2d\n", 2, $tpct, 0, 0, 0, 0;
    printf OUT1 "%11.6f%4.1f%4.1f%4.1f\n", $length, 0, 0, 0, 0;
}

$inpt = <PTIN>;                  # read in the first point for calculations
$fromx = substr($inpt, 59, 18);
$fromy = substr($inpt, 77, 17);
$geox = substr($inpt, 13, 28);
$geoy = substr($inpt, 31, 28);
if($rdtype == 3) {
    printf MERGE "%7d %10.6f %10.6f\n", $tpct, $geox, $geoy;
}

for($i = 1; $i < $verts; $i++) {  # loop for number of vertices
    $inpt = <PTIN>;
    $tox = substr($inpt, 59, 18);
    $toy = substr($inpt, 77, 17);

    $deltax = $fromx - $tox;      # calculating distance between the 2
    $deltay = $fromy - $toy;
    $z = (sqrt(($deltax ** 2) + ($deltay ** 2)) / 1000);

    if($rdtype == 3) {
        printf OUT1 "%10.6f %11.6f %13.6f\n", $geox, $geoy, $z;
    }

    $geox = substr($inpt, 13, 28);
    $geoy = substr($inpt, 31, 28);
    $fromx = $tox;
    $fromy = $toy;
}

if($rdtype == 3) {
    printf OUT1 "%10.6f %11.6f %6d.\n", $geox, $geoy, 0;
    printf MERGE "%7d %10.6f %10.6f\n", $tpct, $geox, $geoy;
    print OUT1 "END\n";
    ++$tpct;
}

}
print OUT1 "END\n";
close(PTIN);
close(LNIN);
close(OUT1);

# printing out beginning number for file #2
print RDSTAT "$tpct\n";

```

```

# ##### Now for line 2
# open various files for reading and writing
open(PTIN,"xxtransin1.txt");           # point information in
open(LNIN,"xxtransin2.txt");           # line information in
open(OUT1, ">xxtransout2.txt");        # type 1 roads output file
$blah = 102;
# read information in, format it, and output to the appropriate files
while($inline = <LNIN>) {             # cycle as long as roads remain
    $verts = substr($inline, 5, 6);     # number of vertices
    $rdtype = substr($inline, 22, 1);   # type of road
    $length = substr($inline, 23, 12); # overall length of the road

    $verts *= 1;
    $rdtype *= 1;
    $length /= 1000;
    $blah *= 1;

    # print out the header line information
    if($rdtype == 2) {
        print OUT1 " $blah $verts";
        printf OUT1 "%2d%7d%2d%2d%2d%2d\n", 2, $tpct, 0, 0, 0, 0;
        printf OUT1 "%11.6f%4.1f%4.1f%4.1f%4.1f\n", $length, 0, 0, 0, 0;
    }

    $inpt = <PTIN>;                  # read in the first point for calculations
    $fromx = substr($inpt, 59, 18);
    $fromy = substr($inpt, 77, 17);
    $geox = substr($inpt, 13, 28);
    $geoy = substr($inpt, 31, 28);
    if($rdtype == 2) {
        printf MERGE "%7d %10.6f %10.6f\n", $tpct, $geox, $geoy;
    }

    for($i = 1; $i < $verts; $i++) {  # loop for number of vertices
        $inpt = <PTIN>;
        $tox = substr($inpt, 59, 18);
        $toy = substr($inpt, 77, 17);

        $deltax = $fromx - $tox;      # calculating distance between the 2
        $deltay = $fromy - $toy;
        $z = ($sqrt((($deltax ** 2) + ($deltay ** 2)) / 1000);

        if($rdtype == 2) {
            printf OUT1 "%10.6f %11.6f %13.6f\n", $geox, $geoy, $z;
        }

        $geox = substr($inpt, 13, 28);
        $geoy = substr($inpt, 31, 28);
        $fromx = $tox;
        $fromy = $toy;
    }

    if($rdtype == 2) {
        printf OUT1 "%10.6f %11.6f %6d.\n", $geox, $geoy, 0;
        printf MERGE "%7d %10.6f %10.6f\n", $tpct, $geox, $geoy;
        print OUT1 "END\n";
        ++$tpct;
    }
}

}
print OUT1 "END\n";
close(PTIN);
close(LNIN);
close(OUT1);

# printing out beginning number for file #3
print RDSTAT "$tpct\n";

# ##### Now for roads type 3
# open various files for reading and writing
open(PTIN,"xxtransin1.txt");           # point information in
open(LNIN,"xxtransin2.txt");           # line information in
open(OUT1, ">xxtransout3.txt");        # type 1 roads output file
$blah = 103;
# read information in, format it, and output to the appropriate files
while($inline = <LNIN>) {             # cycle as long as roads remain
    $verts = substr($inline, 5, 6);     # number of vertices
    $rdtype = substr($inline, 22, 1);   # type of road
    $length = substr($inline, 23, 12); # overall length of the road

    $verts *= 1;
    $rdtype *= 1;
    $length /= 1000;
    $blah *= 1;

    # print out the header line information
    if($rdtype == 1) {
        print OUT1 " $blah $verts";
        printf OUT1 "%2d%7d%2d%2d%2d%2d\n", 2, $tpct, 0, 0, 0, 0;
        printf OUT1 "%11.6f%4.1f%4.1f%4.1f%4.1f\n", $length, 0, 0, 0, 0;
    }

    $inpt = <PTIN>;                  # read in the first point for calculations
    $fromx = substr($inpt, 59, 18);
    $fromy = substr($inpt, 77, 17);
    $geox = substr($inpt, 13, 28);
    $geoy = substr($inpt, 31, 28);
    if($rdtype == 1) {
        printf MERGE "%7d %10.6f %10.6f\n", $tpct, $geox, $geoy;
    }

    for($i = 1; $i < $verts; $i++) {  # loop for number of vertices
        $inpt = <PTIN>;
        $tox = substr($inpt, 59, 18);
        $toy = substr($inpt, 77, 17);

        $deltax = $fromx - $tox;      # calculating distance between the 2
        $deltay = $fromy - $toy;
        $z = ($sqrt((($deltax ** 2) + ($deltay ** 2)) / 1000);

        if($rdtype == 1) {
            printf OUT1 "%10.6f %11.6f %13.6f\n", $geox, $geoy, $z;
        }
    }
}

```

```

$geox = substr($inpt, 13, 28);
$geoy = substr($inpt, 31, 28);
$fromx = $tox;
$fromy = $toy;
}

if($rdtype == 1) {
    printf OUT1 "%10.6f %11.6f %6d.\n", $geox, $geoy, 0;
    print MERGE "%7d %10.6f %10.6f\n", $tpct, $geox, $geoy;
    print OUT1 "END\n";
    ++$pct;
}
}

print OUT1 "END\n";
close(PTIN);
close(LNIN);
close(OUT1);
close(MERGE);
close(RDSTAT);

```

## B.6 Hydrology - linear features

### B.6.1 154hydro.aml

```

/* 154hydro.aml
/*
 * Program: 154hydro.aml
 * Purpose: To format the arc features (rivers into
 * files which can be used by CBS 154, and copy these files into
 * the final output directory.
 */
/*
 * Called By: dcwcbs154.aml
 * Calls Made: 154hydro.perl
*/
/*
 * Arguments: none
 * Globals: .dirname - the name of the output directory
 *           .name - trunk name for the output files
 */
/*
 * Inputs: none
 * Outputs: correctly formatted arc files for cbs 154
 */
/*
 * History: 6 May 1997 - Tobi Steinberg
 * steinbet@email.grafenwoehr.army.mil
 * http://199.56.131.202/~tobi/
 * Terrain Simulations - 7th ATC - Grafenwoehr, Germany
 */
/*
&severity &error &routine bail
&echo &on
display 0
&ty [date -VFULL]

&s incov xhydrox

/* Checking on the existence of coverage names needed by the program
&do cov &list %incov%gpt %incov%pt %incov%lpt
    &if [exists %cov% -cover] &then kill %cov% all
&end

/* Deal with the coverage with all three component sizes together; split
/* later in the perl script/* Adding items for x and y coordinates in
both geographic and meters
/* Getting the meters values from Lambert since I already have a set of
/* coverages in that projection and can easily determine the parameters
arcpoint %incov%g %incov%gpt line %incov%g#
addxy %incov%gpt point
tables
    sel %incov%gpt.pat
    alter x-coord
        geox
        ~
        ~
        ~
        ~
    alter y-coord
        geoy
        ~
        ~

```

```

~
~
quit

/* Projecting the point coverage into lambertcc
&describe %incov%
project cover %incov%gpt %incov%lpt
  output
    projection lambert
  units meters
  spheroid wgs84
  datum wgs84
parameters
%prj$sp1%
%prj$sp2%
%prj$cm%
~
~
~
end

countvertices %incov%g line
&sys rm xxhydroin?.txt
addxy %incov%lpt point
tables
  sel %incov%lpt.pat
  unload xxhydroin1.txt %incov%g# geox geoy x-coord y-coord columnar
junk.txt
  sel %incov%g.aat
  unload xxhydroin2.txt %incov%g# vertices dnlntype dnlnstat length
columnar junk.txt
  &sys rm junk.txt
quit

/* Calling the perl script to format the file
&sys /data2/cbs/154/aml/154hydro.perl
kill %incov%g all
kill %incov%gpt all
kill %incov%lpt all
&sys rm xxhydroin?.txt

display 9999
&ty [date -VFULL]
&return
*****  

/* bail routine
*****  

&routine bail
&severity &error &ignore
&severity &warning &ignore
&type An error has occured in 154hydro.aml
&type Bailing out of 154hydro.aml
&return; &return &error

```

## B.6.2 154hydro.perl

```

#! /usr/local/bin/perl

# -----
#   Program: 154hydro.perl
#   Purpose: to format a text file output by 154hydro.aml into the
#             correct format for cbs 154. Used for rivers.
# -----
#   Called By: 154hydro.aml
#   Calls Made: none
# -----
#   Arguments: none
#   Globals: none
# -----
#   Inputs: text file created by 154hydro.aml - point in
xxhydroin1.txt
#           lines in xxhydroin2.txt
#   Outputs: correctly formatted text file for cbs 154
# -----
#   History: 6 May 1997 - Tobi Steinberg
#             steinbet@email.grafenwoehr.army.mil
#             http://199.56.131.202/~tobi/
#             Terrain Simulations - 7th ATC - Grafenwoehr, Germany
# -----
# Setting counts of files for rivers of three different types
$pct = 1;

$blah = 201;

# Repeat this entire section three times. Once for line 1, then for 2,
then for 3
# open various files for reading and writing
open(PTIN,"xxhydroin1.txt");      # point information in
open(LNIN,"xxhydroin2.txt");      # line information in
open(OUT1, ">xxhydroout1.txt");    # type 1 rivers output file

# creating this file to be used later by 154bridges2.perl
open(HYSTAT, ">xxhydstat.txt");
$cbsid = 200000 + $pct;
print HYSTAT "$cbsid\n";

# read information in, format it, and output to the appropriate files
while($inline = <LNIN>) {          # cycle as long as rivers remain
  $verts = substr($inline, 5, 6);    # number of vertices
  $dntype = substr($inline, 22, 1);  # type of river
  $dnstat = substr($inline, 23, 12); # status of river
  $length = substr($inline, 35, 18); # overall length of the river

  $verts *= 1;
  $dntype *= 1;
  $dnstat *= 1;
  $length /= 1000;
  $blah *= 1;
}

```

```

# print out the header line information
if($dnstat == 2) {
    print OUT1 " $blah $verts";
    printf OUT1 "%2d%7d%2d%2d%2d\n", 2, (200000 + $tpct), 0, 0, 0,
0;
    printf OUT1 "%11.6f%4.1f%4.1f%4.1f%4.1f\n", $length, 0, 0, 0, 0;
}

$inpt = <PTIN>;           # read in the first point for calculations
$fromx = substr($inpt, 59, 18);
$fromy = substr($inpt, 77, 17);
$geox = substr($inpt, 13, 28);
$geoy = substr($inpt, 31, 28);

for($i = 1; $i < $verts; $i++) {  # loop for number of vertices
    $inpt = <PTIN>;
    $tox = substr($inpt, 59, 18);
    $toy = substr($inpt, 77, 17);

    $deltax = $fromx - $tox;      # calculating distance between the 2
    $deltay = $fromy - $toy;
    $z = (sqrt(($deltax ** 2) + ($deltay ** 2)) / 1000);

    if($dnstat == 2) {
        printf OUT1 "%10.6f %11.6f %13.6f\n", $geox, $geoy, $z;
    }

    $geox = substr($inpt, 13, 28);
    $geoy = substr($inpt, 31, 28);
    $fromx = $tox;
    $fromy = $toy;
}

if($dnstat == 2) {
    printf OUT1 "%10.6f %11.6f %6d.\n", $geox, $geoy, 0;
    print OUT1 "END\n";
    ++$tpct;
}

print OUT1 "END\n";
close(PTIN);
close(LNIN);
close(OUT1);

$blah = 202;

# printint out to the statistics file the number that file #2 begins with
$cbsid = 200000 + $tpct;
print HYSTAT "$cbsid\n";

# ##### for medium sized rivers
# open various files for reading and writing
open(PTIN, "xxhydroin1.txt");          # point information in
open(LNIN, "xxhydroin2.txt");          # line information in
open(OUT1, ">xxhydroout2.txt");        # type 1 rivers output file
# read information in, format it, and output to the appropriate files

```

```

while($inline = <LNIN>) {           # cycle as long as rivers remain
    $verts = substr($inline, 5, 6);    # number of vertices
    $dnctype = substr($inline, 22, 1);  # type of river
    $dnstat = substr($inline, 23, 12); # status of river
    $length = substr($inline, 35, 18); # overall length of the river

    $verts *= 1;
    $dnctype *= 1;
    $dnstat *= 1;
    $length /= 1000;
    $blah *= 1;

    if($dnstat == 5) {
        $dnctype = 4;
    }

    # print out the header line information
    if($dnctype == 4) {
        print OUT1 " $blah $verts";
        printf OUT1 "%2d%7d%2d%2d%2d\n", 2, (200000 + $tpct), 0, 0, 0,
0;
        printf OUT1 "%11.6f%4.1f%4.1f%4.1f%4.1f\n", $length, 0, 0, 0, 0;
    }

    $inpt = <PTIN>;           # read in the first point for calculations
    $fromx = substr($inpt, 59, 18);
    $fromy = substr($inpt, 77, 17);
    $geox = substr($inpt, 13, 28);
    $geoy = substr($inpt, 31, 28);

    for($i = 1; $i < $verts; $i++) {  # loop for number of vertices
        $inpt = <PTIN>;
        $tox = substr($inpt, 59, 18);
        $toy = substr($inpt, 77, 17);

        $deltax = $fromx - $tox;      # calculating distance between the 2
        $deltay = $fromy - $toy;
        $z = (sqrt(($deltax ** 2) + ($deltay ** 2)) / 1000);

        if($dnctype == 4) {
            printf OUT1 "%10.6f %11.6f %13.6f\n", $geox, $geoy, $z;
        }

        $geox = substr($inpt, 13, 28);
        $geoy = substr($inpt, 31, 28);
        $fromx = $tox;
        $fromy = $toy;
    }

    if($dnctype == 4) {
        printf OUT1 "%10.6f %11.6f %6d.\n", $geox, $geoy, 0;
        print OUT1 "END\n";
        ++$tpct;
    }
}

print OUT1 "END\n";
close(PTIN);

```

```

close(LNIN);
close(OUT1);

$blah = 203;

# printint out to the statistics file the number that file #3 begins with
$cbsid = 200000 + $tpct;
print HYSTAT "$cbsid\n";

# ##### now for small rivers
# open various files for reading and writing
open(PTIN, "xxhydroin1.txt");           # point information in
open(LNIN, "xxhydroin2.txt");           # line information in
open(OUT1, ">xxhydroout3.txt");         # type 1 rivers output file
# read information in, format it, and output to the appropriate files
while($inline = <LNIN>) {               # cycle as long as rivers remain
    $verts = substr($inline, 5, 6);       # number of vertices
    $dntype = substr($inline, 22, 1);     # type of river
    $dnstat = substr($inline, 23, 12);    # status of river
    $length = substr($inline, 35, 18);    # overall length of the river

    $verts *= 1;
    $dntype *= 1;
    $dnstat *= 1;
    $length /= 1000;
    $blah *= 1;

    # print out the header line information
    if($dnstat == 1) {
        print OUT1 " $blah $verts";
        printf OUT1 "%2d%7d%2d%2d%2d\n", 2, (200000 + $tpct), 0, 0, 0,
0;
        printf OUT1 "%11.6f%4.1f%4.1f%4.1f%4.1f\n", $length, 0, 0, 0, 0;
    }

    $inpt = <PTIN>;                   # read in the first point for calculations
    $fromx = substr($inpt, 59, 18);
    $fromy = substr($inpt, 77, 17);
    $geox = substr($inpt, 13, 28);
    $geoy = substr($inpt, 31, 28);

    for($i = 1; $i < $verts; $i++) {   # loop for number of vertices
        $inpt = <PTIN>;
        $tox = substr($inpt, 59, 18);
        $toy = substr($inpt, 77, 17);

        $deltax = $fromx - $tox;      # calculating distance between the 2
        $deltay = $fromy - $toy;
        $z = (sqrt(($deltax ** 2) + ($deltay ** 2)) / 1000);

        if($dnstat == 1) {
            printf OUT1 "%10.6f %11.6f %13.6f\n", $geox, $geoy, $z;
        }

        $geox = substr($inpt, 13, 28);
        $geoy = substr($inpt, 31, 28);
        $fromx = $tox;
    }
}

$fromy = $toy;
}

if($dnstat == 1) {
    printf OUT1 "%10.6f %11.6f %6d.\n", $geox, $geoy, 0;
    print OUT1 "END\n";
    ++$tpct;
}

print OUT1 "END\n";
close(PTIN);
close(LNIN);
close(OUT1);
close(HYSTAT);

```

## B.7 Vegetation

### B.7.1 154veg.aml

```
/* 154veg.aml
* -----
/* Program: 154veg.aml
/* Purpose: Create veg polygon coordinate files to be formatted
/* by the perl script
* -----
/* Called By: dcwcbs154.aml
/* Calls Made: 154veg.perl
* -----
/* Arguments: none
/* Globals: .dirname, .name
* -----
/* History: 3 September 1997 - Tobi Steinberg
/* steinbet@email.grafenwoehr.army.mil
/* http://199.56.131.202/~tobi/
/* Terrain Simulations - 7th ATC - Grafenwoehr, Germany
* -----
&severity &error &routine bail
&echo &on
display 0
&ty [date -VFULL]

/* Checking on the existence of coverage names needed by the program
&do cov &list xvegx
  &if [exists %cov% -cover] &then kill %cov% all
&end&sys rm xxvegin?.txt
build xvegx

/* projecting the coverage into geographic - it needs to start out as
/* lcc!!!
project cover xvegx xvegxg
  output
    projection geographic
  units dd
  spheroid wgs84
  datum wgs84
  parameters
end

countvertices xvegx poly
&s tester .TRUE. /* This will be false if the item already exists

build xvegxg
ae
  /* testing to see if pppyytype already exists
  &s items [show item xvegx.pat]
  &do item &list %items%
    &if %item% eq PPPYTYPE &then &s tester .FALSE.
  &end
quit

&if %tester% &then additem xvegx.pat xvegx.pat pppyytype 4 12 B
```

```
tables
  /* Outputting other information about each polygon
  /* changing the values of pppyytype to fulfill 154 format requirements
sel xvegx.pat
alter
area
~
20
~
~
~
sel xvegx.pat
calc pppyytype = 303
resel area gt 0
sort area (D)
unload xxvegin3.txt xvegx# vertices area pppyytype perimeter xvegx-id
columnar junk.txt
  sort xvegx#
quit

/* Outputting polygon coordinates in lambert meters
arcross xvegx xxvegin1.txt utm xvegx-id # poly
/* Outputting polygon coordinates in geographic decimal degrees
arcross xvegxg xxvegin2.txt latlong xvegxg-id # poly
&sys rm junk.txt

/* Call perl script to do the text file formatting
&sys /data2/cbs/154/amls/154veg.perl

&ty ****
&ty Copy xxvegout1.txt to the output file directory right away!!!
&ty ****
&sys rm xxvegin*.txt
&sys rm xxvegout5.txt
&sys rm xxperllog.txt

kill xvegxg all

display 9999
&ty [date -VFULL]
&return
*****
/* bail routine
*****
&routine bail
&severity &error &ignore
&severity &warning &ignore
&type An error has occurred in 154veg.aml
&type Bailing out of 154veg.aml
&return; &return &error
```

## B.7.2 154veg.perl

```

#!/usr/local/bin/perl

# -----
#     Program: 154veg.perl
#     Purpose: to format a text file output by 154veg.aml into the
#             correct format for cbs 154. Used for veg.
# -----
#     Called By: 154veg.aml
#     Calls Made: none
# -----
#     Arguments: none
#     Globals: none
# -----
#     Inputs: text file created by 154veg.aml - point in xxvegin1.txt
#             lines in xxvegin2.txt
#     Outputs: correctly formatted text file for cbs 154
# -----
#     History: 6 May 1997 - Tobi Steinberg
#               steinbet@email.grafenwoehr.army.mil
#               http://199.56.131.202/~tobi/
#     Terrain Simulations - 7th ATC - Grafenwoehr, Germany
#     More work 12 August 1997, fixing.
# -----

# Setting counts of files for veg of three different types
$tpct = 1;

$blah = 103;

# open various files for reading and writing
open(IN3, "xxvegin3.txt");      # other information about each poly
open(OUT1, ">xxvegout1.txt");    # output file for final text
open(PERLOG, ">xxperlog.txt");   # temporary debugging file

# reformat the geo and meters text files so that they are one file, with
# side by side coords
open(OUT5, ">xxvegout5.txt");    # temporary text file used only in the
script

print "Creating temporary file, xxvegout5.txt...\n";
print "\n";
while($inline3 = <IN3>) {
    $vegid = substr($inline3, 55, 5) * 1; # the xvegx# for this poly
    $verts = substr($inline3, 5, 6) * 1;   # the number of vertices for
this poly

    # having to do this inside the loop because they coordinate lists
    # aren't always in order

    open(LCC, "xxvegin1.txt");        # input coordinates in lambert meters
    open(GEO, "xxvegin2.txt");        # input coordinates in decimal degrees

    # finding the location in the coordinate lists for this polygon
    # finding for decimal degrees

```

```

$truth = 1;
while ($truth == 1) {
    $geoin = <GEO>;
    $tempct = substr($geoin, 14, 5) * 1;
    if ($tempct == $vegid) {
        $truth = 0;
        $tempvpt = substr($geoin, 50);
        chop($tempvpt);
        $tempvpt = $tempvpt * 1;
    }
}

# finding for lambert meters
$truth = 1;
while ($truth == 1) {
    $lccin = <LCC>;
    $tempct = substr($lccin, 8, 17) * 1;
    if ($tempct == $vegid) {
        $truth = 0;
    }
}

print OUT5 "$vegid\n";
# extracting the coordinates and outputting them to the OUT5 file
for($i = 1; $i < $verts; $i++) { # loop for the number of vertices
    $lccin = <LCC>;
    $geoin = <GEO>;
    chop($lccin);
    chop($geoin);
    if ($i == 1) {
        $initlcc = $lccin;
        $initgeo = $geoin;
    }
    print OUT5 "$lccin $geoin\n"; # print out all four coord to the
output file
}
print OUT5 "$initlcc $initgeo\n";

close(LCC);
close(GEO);

}
close(OUT5);
close(IN3);

$seqcount = 300001;
open(INFO, "xxvegin3.txt");
open(MTIN, "xxvegout5.txt");

print "Reformatting as necessary...\n";
print "\n";
# read information in, format it, and output to the appropriate files
while($inline = <INFO>) { # cycle as long as veg remain
    $urbnum = substr($inline, 0, 5); # read out the xvegx# from the arc
coverage
    $urbnum *= 1;
    print PERLOG "$urbnum\n";
}

```

```

if($urbnum) {
    $verts = substr($inline, 5, 6);      # number of vertices
    $area = substr($inline, 11, 20);    # area of this veg polygon
    $urtype = substr($inline, 40, 3);   # type: 401, 402, or 403
    $perim = substr($inline, 43, 12);

    $verts *= 1;                      # reminding all these variable
    that they're numbers
    $area /= 1000;
    $urtype *= 1;
    $perim /= 1000;

    # print out the header line information
    print OUT1 "$urtype $verts 2 $seqcount 0 0 0 0\n";
    ++$seqcount;
    printf OUT1 "%13.6f %19.6f %4.1f %4.1f %4.1f\n", $perim, $area, 0,
0, 0;
    print PERLOG "Working on line #$urbnum with $verts vertices\n";

    $inpt = <MTIN>;                  # read & discard the xvegx-id
    $inpt = <MTIN>;                  # read in the first point for
calculations
    $fromx = substr($inpt, 0, 11) * 1;
    $fromy = substr($inpt, 11, 11) * 1;
    $geox = substr($inpt, 31, 10) * 1;
    $geoy = substr($inpt, 41, 10) * 1;
    print PERLOG "fromx - $fromx fromy - $fromy geox - $geox geoy -
$geoy\n";

    for($i = 1; $i < $verts; $i++) {    # loop for number of vertices
        $inpt = <MTIN>;
        $tox = substr($inpt, 0, 11) * 1;
        $toy = substr($inpt, 11, 11) * 1;

        $deltax = $fromx - $tox;      # calculating distance between the
2
        $deltay = $fromy - $toy;
        $z = sqrt(abs(($deltax ** 2) + ($deltay ** 2))) / 1000;

        printf OUT1 "%11.6f %12.6f %14.6f\n", $geox, $geoy, $z;
        $fromx = $tox;    # reset fromx and fromy for the next time
around
        $fromy = $toy;
        $geox = substr($inpt, 31, 10) * 1; # reset geox and geoy
        $geoy = substr($inpt, 41, 10) * 1;
        # print PERLOG "fromx - $fromx fromy - $fromy geox - $geox geoy
- $geoy #2\n";
    }

    # printing last line for poly, and word 'END' at the end
    print PERLOG "End of line $urbnum at vertex $i\n";
    printf OUT1 "%11.6f %12.6f %7d.\n", $geox, $geoy, 0;
    print OUT1 "END\n";
} else {
    close(MTIN);
}
    print OUT1 "END\n";
}
print OUT1 "END\n";
close(OUT1);
close(PERLOG);
print "Done with PERL script\n";

```

## B.8 Urban

### B.8.1 154urban.aml

```
/* 154urban.aml
* -----
*     Program: 154urban.aml
*     Purpose: Create urban polygon coordinate files to be formatted
*             by the perl script
* -----
*     Called By: dcwcbs154.aml
*     Calls Made: 154urban.perl
* -----
*     Arguments: none
*     Globals: .dirname, .name
*     Locals:
* -----
*     Inputs:
*     Outputs:
* -----
*     History: date - Tobi Steinberg
*             steinbet@email.grafenwoehr.army.mil
*             http://199.56.131.202/~tobi/
*     Terrain Simulations - 7th ATC - Grafenwoehr, Germany
* -----
&severity &error &routine bail
&echo &on
display 0
&ty [date -VFULL]

/* Checking on the existence of coverage names needed by the program
&do cov &list
    &if [exists %cov% -cover] &then kill %cov% all
&end

&sys rm xxurbanin?.txt
build xurbanx

/* Outputting other information about each polygon
countvertices xurbanx poly
tables
    /* changing the values of pppotype to fulfill 154 format requirements
sel xurbanx.pat
alter
area
~
20
~
~
~
~
resel pppotype gt 0
calc pppotype = 403
resel area lt 230000
calc pppotype = 402
resel area lt 50000
calc pppotype = 401
```

```
sel xurbanx.pat
resel pppotype gt 0
sort area (D)
unload xxurbanin3.txt xurbanx# vertices area pppotype perimeter
xurbanx-id columnar junk.txt
    sort xurbanx#
quit

/* Outputting polygon coordinates in lambert meters
arcross xurbanx xxurbanin1.txt utm xurbanx-id # poly

/* Outputting polygon coordinates in geographic decimal degrees
arcross xurbanxg xxurbanin2.txt latlong xurbanxg-id # poly

&sys rm junk.txt

/* Call perl script to do the text file formatting
&sys /data2/cbs/154/amls/154urban.perl

/* Move and reformat the names of the output file
&sys cp xxurbanout1.txt %.dirname%/.name%_urban.txt
&if [exists %.dirname%/.name%_urban.txt -file] &then
    &s junk [delete xxurbanout1.txt -file]

&sys rm xxurbanin*.txt
&sys rm xxurbanout5.txt

display 9999
&ty [date -VFULL]
&return
*****
/* bail routine
*****
&routine bail
&severity &error &ignore
&severity &warning &ignore
&type An error has occurred in 154urban.aml
&type Bailing out of 154urban.aml
&return; &return &error
```

## B.8.2 154urban.perl

```

#!/usr/local/bin/perl

# -----
#     Program: 154urban.perl
#     Purpose: to format a text file output by 154urban.aml into the
#             correct format for cbs 154. Used for urban.
# -----
#     Called By: 154urban.aml
#     Calls Made: none
# -----
#     Inputs: text file created by 154urban.aml - point in
xxurbanin1.txt
#             lines in xxurbanin2.txt
#     Outputs: correctly formatted text file for cbs 154
# -----
#     History: 6 May 1997 - Tobi Steinberg
#             steinbet@email.grafenwoehr.army.mil
#             http://199.56.131.202/~tobi/
#     Terrain Simulations - 7th ATC - Grafenwoehr, Germany
#     More work 12 August 1997, fixing.
# -----
# Setting counts of files for urban of three different types
$tpct = 1;

$blah = 103;

# open various files for reading and writing
open(IN3, "xxurbanin3.txt");          # other information about each poly
open(OUT1, ">xxurbanout1.txt");       # output file for final text
open(PERLOG, ">xxperlog.txt");        # temporary debugging file

# reformat the geo and meters text files so that they are one file, with
# side by side coords
open(OUT5, ">xxurbanout5.txt");       # temporary text file used only in
the script

print "Creating temporary file, xxurbanout5.txt...\n";
print "\n";
while($inline3 = <IN3>) {
    $urbanid = substr($inline3, 55, 5) * 1; # the xurbanx# for this poly
    $verts = substr($inline3, 5, 6) * 1;    # the number of vertices for
this poly

    # having to do this inside the loop because they coordinate lists
    # aren't always in order

    open(LCC, "xxurbanin1.txt");          # input coordinates in lambert
meters
    open(GEO, "xxurbanin2.txt");          # input coordinates in decimal
degrees

    # finding the location in the coordinate lists for this polygon
    # finding for decimal degrees

```

```

$truth = 1;
while ($truth == 1) {
    $geoin = <GEO>;
    $tempct = substr($geoin, 14, 5) * 1;
    if ($tempct == $urbanid) {
        $truth = 0;
        $tempvpt = substr($geoin, 50);
        chop($tempvpt);
        $tempvpt = $tempvpt * 1;
    }
}

# finding for lambert meters
$truth = 1;
while ($truth == 1) {
    $lccin = <LCC>;
    $tempct = substr($lccin, 8, 17) * 1;
    if ($tempct == $urbanid) {
        $truth = 0;
    }
}

print OUT5 "$urbanid\n";
# extracting the coordinates and outputting them to the OUT5 file
for($i = 1; $i < $verts; $i++) { # loop for the number of vertices
    $lccin = <LCC>;
    $geoin = <GEO>;
    chop($lccin);
    chop($geoin);
    if ($i == 1) {
        $initlcc = $lccin;
        $initgeo = $geoin;
    }
    print OUT5 "$lccin $geoin\n"; # print out all four coord to the
output file
}
print OUT5 "$initlcc $initgeo\n";

close(LCC);
close(GEO);

}
close(OUT5);
close(IN3);

$seqcount = 400001;
open(INFO, "xxurbanin3.txt");
open(MTIN, "xxurbanout5.txt");

print "Reformatting as necessary...\n";
print "\n";
# read information in, format it, and output to the appropriate files
while($inline = <INFO>) { # cycle as long as urban remain
    $urbnum = substr($inline, 0, 5); # read out the xurbanx# from the arc
coverage
    $urbnum *= 1;
    print PERLOG "$urbnum\n";
}

```

```

if($urbnum) {
    $verts = substr($inline, 5, 6);      # number of vertices
    $area = substr($inline, 11, 20);    # area of this urban polygon
    $urtype = substr($inline, 40, 3);   # type: 401, 402, or 403
    $perim = substr($inline, 43, 12);

    $verts *= 1;                      # reminding all these variable
    that they're numbers
    $area /= 1000;
    $urtype *= 1;
    $perim /= 1000;

    # print out the header line information
    print OUT1 "$urtype $verts 2 $seqcount 0 0 0 0\n";
    ++$seqcount;
    printf OUT1 "%13.6f %19.6f %4.1f %4.1f %4.1f\n", $perim, $area, 0,
0, 0;
    print PERLOG "Working on line #$urbnum with $verts vertices\n";

    $inpt = <MTIN>;                  # read & discard the xurbanx-id
    $inpt = <MTIN>;                  # read in the first point for
calculations
    $fromx = substr($inpt, 0, 11) * 1;
    $fromy = substr($inpt, 11, 11) * 1;
    $geox = substr($inpt, 31, 10) * 1;
    $geoy = substr($inpt, 41, 10) * 1;
    print PERLOG "fromx - $fromx fromy - $fromy geox - $geox geoy -
$geoy\n";

    for($i = 1; $i < $verts; $i++) {    # loop for number of vertices
        $inpt = <MTIN>;
        $tox = substr($inpt, 0, 11) * 1;
        $toy = substr($inpt, 11, 11) * 1;

        $deltax = $fromx - $tox;      # calculating distance between the
2
        $deltay = $fromy - $toy;
        $z = sqrt(abs(($deltax ** 2) + ($deltay ** 2))) / 1000;

        printf OUT1 "%11.6f %12.6f %14.6f\n", $geox, $geoy, $z;
        $fromx = $tox;    # reset fromx and fromy for the next time
around
        $fromy = $toy;
        $geox = substr($inpt, 31, 10) * 1; # reset geox and geoy
        $geoy = substr($inpt, 41, 10) * 1;
        # print PERLOG "fromx - $fromx fromy - $fromy geox - $geox geoy
- $geoy #2\n";
    }

    # printing last line for poly, and word 'END' at the end
    print PERLOG "End of line $urbnum at vertex $i\n";
    printf OUT1 "%11.6f %12.6f %7d.\n", $geox, $geoy, 0;
    print OUT1 "END\n";
} else {
    close(MTIN);
}
    print OUT1 "END\n";
}
print OUT1 "END\n";
close(OUT1);
close(PERLOG);
print "Done with PERL script\n";

```

## B.9 Trafficability

### B.9.1 154traf.aml

```
/* 154traf.aml
/*
-----  
* Program: 154traf.aml  
* Purpose: Create traf polygon coordinate files to be formatted  
* by the perl script  
* -----  
* Called By: dcwcbs154.aml  
* Calls Made: 154traf.perl  
* -----  
* Arguments: none  
* Globals: .dirname, .name  
* Locals: lots  
* -----  
* Inputs: xxtrafgrid, a DTED grid over the whole area  
* Outputs: %.name%_traf.txt  
* -----  
* History: 3 September 1997 - Tobi Steinberg  
* steinbet@email.grafenwoehr.army.mil  
* http://199.56.131.202/~tobi/  
* Terrain Simulations - 7th ATC - Grafenwoehr, Germany  
* -----  
  
&severity &error &routine bail  
&echo &on  
display 0  
&ty [date -VFULL]  
  
/* Checking on the existence of coverage names needed by the program  
&do cov &list xtrafxg xxslopepoly xxslope7 xtrafx  
  &if [exists %cov% -cover] &then kill %cov% all  
&end  
&do grid &list xxtrafgrid2 xxslope xxslope2 xxslope3 xxslope4 ~  
xxslope5 xxslope6 xxslope7  
  &if [exists %grid% -grid] &then kill %grid% all  
&end  
  
/* use the grid created in 154tin.aml to calculate trafficability regions  
/* first calculate and reclass based on slope  
/* being careful to delete as we go because this file could be really big  
grid  
  /* Calculating the slope of each cell  
  xxslope = int(slope(xxgrid15, PERCENTRISE))  
  /* kill xxgrid15 all  
  
  /* Grouping the cells into slope categories  
  xxslope2 = con(xxslope <= 15, 10, xxslope)  
    kill xxslope all  
  xxslope3 = con(xxslope2 > 45, 50, xxslope2)  
    kill xxslope2 all  
  xxslope4 = con(xxslope3 > 15, con(xxslope3 <= 25, 20, xxslope3),  
xxslope3)  
    kill xxslope3 all
```

```
  xxslope5 = con(xxslope4 > 25, con(xxslope4 <= 35, 30, xxslope4),  
xxslope4)  
    kill xxslope4 all  
  xxslope6 = con(xxslope5 > 45, con(xxslope5 <= 45, 40, xxslope5),  
xxslope5)  
    kill xxslope5 all  
  xxslope7 = xxslope6 / 10  
    kill xxslope6 all  
  
  /* Getting rid of small patches of odd cells; works better doing it  
this  
  /* way than doing it just once with a bigger window  
  xxslope6 = focalmedian(xxslope7, RECTANGLE, 25, 25, DATA)  
    kill xxslope7 all  
  xxslope7 = focalmedian(xxslope6, RECTANGLE, 25, 25, DATA)  
quit  
  
gridpoly xxslope7 xxslopepoly 18  
kill xxslope7 all  
rename xxslopepoly xtrafx  
  
/* next combine with the oceans & lakes coverage to update - in these  
coverages  
/* there needs to be an item called grid-code with the value of 6  
  
&sys rm xxtrafin?.txt  
tolerance xtrafx nodesnap 1  
tolerance xtrafx weed 1  
  
/* projecting the coverage into geographic - it needs to start out as  
/* lcc!!!  
project cover xtrafx xtrafxg  
  output  
  projection geographic  
  units dd  
  spheroid wgs84  
  datum wgs84  
  parameters  
end  
  
countvertices xtrafx poly  
&s tester .TRUE. /* This will be false if the item already exists  
  
build xtrafxg  
ae  
  /* testing to see if pppytype already exists  
  &s items [show item xtrafx.pat]  
  &do item &list %items%  
    &if %item% eq PPPYTYPE &then &s tester .FALSE.  
  &end  
quit  
  
&if %tester% &then additem xtrafx.pat xtrafx.pat pppytype 4 12 B  
  
tables  
  /* Outputting other information about each polygon  
  /* changing the values of pppytype to fulfill 154 format requirements
```

```

sel xtrafx.pat
alter
area
~
20
~
~
~
sel xtrafx.pat
resel grid-code = 1
calc pppotype = 501
sel xtrafx.pat
resel grid-code = 2
calc pppotype = 502
sel xtrafx.pat
resel grid-code = 3
calc pppotype = 503
sel xtrafx.pat
resel grid-code = 4
calc pppotype = 504
sel xtrafx.pat
resel grid-code = 5
calc pppotype = 505
sel xtrafx.pat
resel grid-code = 6
calc pppotype = 506
sel xtrafx.pat
resel area gt 0
sort area (D)
unload xxtrafin3.txt xtrafx# vertices area pppotype perimeter xtrafx-
id columnar junk.txt
sort xtrafx#
quit

/* Outputting polygon coordinates in lambert meters
arcmiss xtrafx xxtrafin1.txt utm xtrafx-id # poly

/* Outputting polygon coordinates in geographic decimal degrees
arcmiss xtrafxg xxtrafin2.txt latlong xtrafxg-id # poly

&sys rm junk.txt

/* Call perl script to do the text file formatting
&sys /data2/cbs/154/amls/154traf.perl

&ty ****
&ty Copy xxtrafout1.txt to the output file directory right away!!!
&ty ****

&sys rm xxtrafin*.txt
&sys rm xxtrafout5.txt
&sys rm xxperlog.txt

kill xtrafxg all

display 9999

```

```

&ty [date -VFULL]
&return
*****
/* bail routine
*****
&routine bail
&severity &error &ignore
&severity &warning &ignore
&type An error has occured in 154traf.aml
&type Bailing out of 154traf.aml
&return; &return &error

```

## B.9.2 154traf.perl

```

#!/usr/local/bin/perl

# -----
#     Program: 154traf.perl
#     Purpose: to format a text file output by 154traf.aml into the
#             correct format for cbs 154. Used for traf.
# -----
#     Called By: 154traf.aml
#     Calls Made: none
# -----
#     Arguments: none
#     Globals: none
#     Locals:
# -----
#     Inputs: text file created by 154traf.aml - point in xxtrafin1.txt
#             lines in xxtrafin2.txt
#     Outputs: correctly formatted text file for cbs 154
# -----
#     History: 6 May 1997 - Tobi Steinberg
#               steinbet@email.grafenwoehr.army.mil
#               http://199.56.131.202/~tobi/
#     Terrain Simulations - 7th ATC - Grafenwoehr, Germany
#     More work 12 August 1997, fixing.
# -----

# Setting counts of files for traf of three different types
$tpct = 1;

$blah = 103;

# open various files for reading and writing
open(IN3, "xxtrafin3.txt");          # other information about each poly
open(OUT1, ">xxtrafout1.txt");       # output file for final text
open(PERLOG, ">xxperlog.txt");       # temporary debugging file

# reformat the geo and meters text files so that they are one file, with
# side by side coords
open(OUT5, ">xxtrafout5.txt");       # temporary text file used only in
the script

print "Creating temporary file, xxtrafout5.txt...\n";
print "\n";
while($inline3 = <IN3>) {
    $trafid = substr($inline3, 55, 5) * 1; # the xtrafx# for this poly
    $verts = substr($inline3, 5, 6) * 1;   # the number of vertices for
this poly

    # having to do this inside the loop because they coordinate lists
    # aren't always in order

    open(LCC, "xxtrafin1.txt");          # input coordinates in lambert meters
    open(GEO, "xxtrafin2.txt");          # input coordinates in decimal
degrees

```

```

# finding the location in the coordinate lists for this polygon
# finding for decimal degrees
$truth = 1;
while ($truth == 1) {
    $geoin = <GEO>;
    $tempct = substr($geoin, 14, 5) * 1;
    if ($tempct == $trafid) {
        $truth = 0;
        $tempvpt = substr($geoin, 50);
        chop($tempvpt);
        $tempvpt = $tempvpt * 1;
    }
}

# finding for lambert meters
$truth = 1;
while ($truth == 1) {
    $lccin = <LCC>;
    $tempct = substr($lccin, 8, 17) * 1;
    if ($tempct == $trafid) {
        $truth = 0;
    }
}

print OUT5 "$trafid\n";
# extracting the coordinates and outputting them to the OUT5 file
for($i = 1; $i < $verts; $i++) { # loop for the number of vertices
    $lccin = <LCC>;
    $geoin = <GEO>;
    chop($lccin);
    chop($geoin);
    if ($i == 1) {
        $initlcc = $lccin;
        $initgeo = $geoin;
    }
    print OUT5 "$lccin $geoin\n"; # print out all four coord to the
output file
}
print OUT5 "$initlcc $initgeo\n";

close(LCC);
close(GEO);

}
close(OUT5);
close(IN3);

$seqcount = 500001;
open(INFO, "xxtrafin3.txt");
open(MTIN, "xxtrafout5.txt");

print "Reformatting as necessary...\n";
print "\n";
# read information in, format it, and output to the appropriate files
while($inline = <INFO>) { # cycle as long as traf remains
    $urbnum = substr($inline, 0, 5); # read out the xtrafx# from the arc
coverage

```

```

$urbnum *= 1;
print PERLOG "$urbnum\n";
if($urbnum) {
    $verts = substr($inline, 5, 6);      # number of vertices
    $area = substr($inline, 11, 20);    # area of this traf polygon
    $urtype = substr($inline, 40, 3);   # type: 401, 402, or 403
    $perim = substr($inline, 43, 12);

    $verts *= 1;                      # reminding all these variable
that they're numbers
    $area /= 1000;
    $urtype *= 1;
    $perim /= 1000;

    # print out the header line information
    print OUT1 " $urtype $verts 2 $seqcount 0 0 0 0\n";
    ++$seqcount;
    printf OUT1 "%13.6f %19.6f %4.1f %4.1f %4.1f\n", $perim, $area, 0,
0, 0;
    print PERLOG "Working on line #$urbnum with $verts vertices\n";

    $inpt = <MTIN>;                  # read & discard the xtrafx-id
    $inpt = <MTIN>;                  # read in the first point for
calculations
    $fromx = substr($inpt, 0, 11) * 1;
    $fromy = substr($inpt, 11, 11) * 1;
    $geox = substr($inpt, 31, 10) * 1;
    $geoy = substr($inpt, 41, 10) * 1;
    print PERLOG "fromx - $fromx fromy - $fromy geox - $geox geoy -
$geoy\n";

for($i = 1; $i < $verts; $i++) {      # loop for number of vertices
    $inpt = <MTIN>;
    $tox = substr($inpt, 0, 11) * 1;
    $toy = substr($inpt, 11, 11) * 1;

    $deltax = $fromx - $tox;        # calculating distance between the
2
    $deltay = $fromy - $toy;
    $z = sqrt(abs(($deltax ** 2) + ($deltay ** 2))) / 1000;

    printf OUT1 "%11.6f %12.6f %14.6f\n", $geox, $geoy, $z;
    $fromx = $tox;      # reset fromx and fromy for the next time
around
    $fromy = $toy;
    $geox = substr($inpt, 31, 10) * 1; # reset geox and geoy
    $geoy = substr($inpt, 41, 10) * 1;
    # print PERLOG "fromx - $fromx fromy - $fromy geox - $geox geoy
- $geoy #2\n";
}

# printing last line for poly, and word 'END' at the end
print PERLOG "End of line $urbnum at vertex $i\n";
printf OUT1 "%11.6f %12.6f %7d.\n", $geox, $geoy, 0;
print OUT1 "END\n";
}
} else {
    close(MTIN);
    print OUT1 "END\n";
}
print OUT1 "END\n";

close(OUT1);
close(PERLOG);
print "Done with PERL script\n";

```

## B.10 Tins

### B.10.1 154tins.aml

```
/*
 * Program: 154tins.aml
 * Purpose: To prompt for DTED disks, import the data, then
 *           create and thin the TIN, then export it.
 */
/*
 * Called By: started by operator
 * Calls Made: 154tins.perl, 154traj.aml
 */
/*
 * Arguments: none
 * Globals: none
 * Locals:
 *           Currently resampling to 15DS before processing DTED
 */
/*
 * Inputs: xtiles (cover), DTED files from CD as prompted
 *           Also need xhy & xoceanx in the workspace to execute properly!!!
 * Outputs: properly formatted tin text file for CBS 1.5.4
 */
/*
 * History: Started 14 August 1997 - Tobi Sellekaerts
 *           steinbet@email.grafenwoehr.army.mil
 *           Terrain Simulation - 7th ATC - Grafenwoehr, Germany
 */
&severity &error &routine bail
&echo &on
display 0
&ty [date -VFULL]

&if [exists tin[date -tag].wat -file] &then
    &s delwat [delete tin[date -tag].wat]
&watch tin[date -tag].wat

/* Checking on the existence of coverage names needed by the program
&do cov &list xxtempgrid xxtarfgrid xylattice xelevx xxgrid15
    &if [exists %cov% -grid] &then kill %cov% all
&end

&do cov &list xxungentin.txt xxungentin.txt.net
    &if [exists %cov% -file] &then &sys rm %cov%
&end

&do cov &list xhytin
    &if [exists %cov% -tin] &then kill %cov% all
&end

&do cov &list xhyds
    &if [exists %cov% -cover] &then kill %cov% all
&end

/* Find the extents of the playbox and round up or down as necessary
/* Working in units of whole degrees
&describe xtiles
&s xmin %dsc%xmin%
&if %xmin% ne [truncate %xmin%] &then &s xmin [truncate %xmin%]
&s xmax %dsc%xmax%
```

```
&if %xmax% ne [truncate %xmax%] &then &s xmax [calc [truncate %xmax%] +
1]
&s ymin %dsc$ymin%
&if %ymin% ne [truncate %ymin%] &then &s ymin [truncate %ymin%]
&s ymax %dsc$ymax%
&if %ymax% ne [truncate %ymax%] &then &s ymax [calc [truncate %ymax%] +
1]
&s xmax %xmax% - 1
&s ymax %ymax% - 1

/* Creating a text file called xdtedcells.txt that contains the
 * latitude and longitude designations of needed dted cells.
&s totcells 0
&s dtedcell [open xdtedcells.txt os -write]
&ty The following DTED cells will be needed:
&do loop1 = %xmin% &to %xmax%
    &do loop2 = %ymin% &to %ymax%
        &if %loop1% >= 0 &then &do /* if in the eastern hemisphere
            &if [length %loop1%] = 1 &then &s x 0%loop1%
                &else &if [length %loop1%] = 2 &then &s x 0%loop1%
                &else &s x %loop1%
                &s x e%x%
                &s junk [write %dtedcell% %x%]
            &end
        &else &do /* if in the western hemisphere
            &s temp [abs %loop1%]
            &if [length %temp%] = 1 &then &s x 00%temp%
                &else &if [length %temp%] = 2 &then &s x 0%temp%
                &else &s x %temp%
                &s x w%x%
                &s junk [write %dtedcell% %x%]
            &end
        &if %loop2% >= 0 &then &do /* if in the northern hemisphere
            &if [length %loop2%] = 1 &then &s y 0%loop2%
                &else &s y %loop2%
                &s y n%y%
                &s junk [write %dtedcell% %y%]
            &end
        &else &do /* if in the southern hemisphere
            &s temp [abs %loop2%]
            &if [length %temp%] = 1 &then &s y 0%temp%
                &else &s y %temp%
                &s y s%y%
                &s junk [write %dtedcell% %y%]
            &end
        &s totcells [calc %totcells% + 1]
    &end
&end

&ty The program will begin by extracting from the DTED CD
&ty already in the drive (if one exists).
&ty It will then prompts you to insert additional CDs.

&s junk [close %dtedcell%]
&s gooddted [open xgddted.txt os -write]
&s goodtot 0
```

```

/* Reading from the dtedcells text file and extracting needed cells
/* from the CD.
&ty Extracting DTED from CD-ROM...
&s drejcnt 0
&s dtedcell [open xtdtedcells.txt os -read]
&s dtedrej [open xtdtedrej.txt os -write]
&do loop1 = 1 &to %totcells%
  &s x [read %dtedcell% rs]
  &s y [read %dtedcell% rs]
  &if ^ [exists x%x%y%grid -grid] &then &do
    &if [exists /cdrom/cdrom0/dted/%x%/%y%.dt1 -file] &then &do
      dtedgrid /cdrom/cdrom0/dted/%x%/%y%.dt1 x%x%y%grid
      &s junk [write %gooddted% x%x%y%grid]
      &s goodtot [calc %goodtot% + 1]
    &end
  &else &do
    &s drejcnt [calc %drejcnt% + 1]
    &ty Did not find DTED cell %x%y%
    &s junk [write %dtedrej% %x%]
    &s junk [write %dtedrej% %y%]
  &end
  &end
  &else &do
    &s junk [write %gooddted% x%x%y%grid]
    &s goodtot [calc %goodtot% + 1]
  &end
  &end
  &end
  &s junk [close %dtedcell%]
  &s junk [close %dtedrej%]

/* If DTED cells need to be extracted from additional CDs, that is
/* accomplished here.
&if %drejcnt% ne 0 &then &do
  &label addcds
  &s cddone [response 'Is this all the available DTED over this area?'
n]
  &if %cddone% = y &then &goto endcds
  &sys eject
  &ty Please insert the next CD-ROM containing DTED data
  &pause
  &sys cp xtdtedrej.txt xtdtedrej2.txt
  &sys rm xtdtedrej.txt
  &s dtedrej2 [open xtdtedrej2.txt os -write]
  &s dtedrej [open xtdtedrej2.txt os -read]
    &s drejcnt2 %drejcnt%
    &s drejcnt 0
    &do loop1 = 1 &to %drejcnt2%
      &s x [read %dtedrej% rs]
      &s y [read %dtedrej% rs]
      &if ^ [exists x%x%y%grid -grid] &then &do
        &if [exists /cdrom/cdrom0/dted/%x%/%y%.dt1 -file] &then &do
          dtedgrid /cdrom/cdrom0/dted/%x%/%y%.dt1 x%x%y%grid
          &s junk [write %gooddted% x%x%y%grid]
          &s goodtot [calc %goodtot% + 1]
        &end
      &else &do
        &s drejcnt [calc %drejcnt% + 1]

```

```

    &ty Did not find DTED cell %x%y%
    &s junk [write %dtedrej2% %x%]
    &s junk [write %dtedrej2% %y%]
  &end
  &end
  &else &do
    &s junk [write %goodtot% x%x%y%grid]
    &s goodtot [calc %goodtot% + 1]
  &end
  &end
  &end
  &s junk [close %dtedrej%]
  &s junk [close %dtedrej2%]
  &if %drcnt% ne 0 &then &goto addcds
  &label endcds
&end

&s junk [close %goodddted%]

&s dtedcell [open xgddted.txt os -read]
&do loop = 1 &to %goodtot%
  &s cellname [read %dtedcell% rs]

  /* Make a resampled copy for use with the trafficability program
(15ds)
  /* and another for use with the TIN building (300ds)
  &s tempname [substr %cellname% 1 8]
  &if ^ [exists %tempname%_15 -grid] &then &do
    laticeresample %cellname% %tempname%_15
    ~
    ~
    ~
    15
  &end
  &if ^ [exists %tempname%_3c -grid] &then &do
    laticeresample %cellname% %tempname%_3c
    ~
    ~
    ~
    300
  &end
  &end
  &s junk [close %dtedcell%]

/* now we need to incorporate the HY data from DCW in the area
/* where there isn't any DTED. Already created xhy in dcwcbs154
ae
  ec xhy
  ef arc
  sel hylntype ne 1
  &if [show number selected] ne 0 &then delete
  save
quit

/* Dealing with everything in Decimal Seconds, so xhy must be
/* projected
project cover xhy xhyds
  output

```

```

projection geographic
units ds
spheroid wgs84
parameters
end

/* checking for the existence of oceans in the database
/* If so, use them to ensure flat oceans
&describe xoceanx
&if %dsc$polygons% gt 1 &then &s oceanpoly .TRUE.
&else &s oceanpoly .FALSE.

/* Creating 300 decimal second grids to be used in the TIN
/* to increase accuracy
&s dtedcell [open xgddted.txt os -read]
&do loop = 1 &to [calc %goodtot% - 1]
  &s cellname [read %dtedcell% rs]
  &s cellname [substr %cellname% 1 8]
  &if ^ [exists %cellname%3pt -cover] &then
    gridpoint %cellname%_3c %cellname%3pt spot
  &end
  &s junk [close %dtedcell%]

/* converting the extents of the playbox into decimal seconds
&s xmax2 [calc %xmax% * 3600]
&s ymax2 [calc %ymax% * 3600]
&s xmin2 [calc %xmin% * 3600]
&s ymin2 [calc %ymin% * 3600]

/* Creating a point coverage with the four corners of the tin area
&if [exists xxtinpt -cover] &then kill xxtinpt all
create xxtinpt
tables
  sel xxtinpt.tic
  add
    1; %xmin2%; %ymin2%
    2; %xmin2%; %ymax2%
    3; %xmax2%; %ymin2%
    4; %xmax2%; %ymax2%
  ~
quit
ae
coordinates keyboard
ec xxtinpt
ef arc
add
  2, %xmin2%, %ymin2%
  1, %xmin2%, %ymax2%
  1, %xmax2%, %ymax2%
  1, %xmax2%, %ymin2%
  2, %xmin2%, %ymin2%
  9
  save
  sel all
  densify 3600
  save
quit

```

```

projectdefine cover xxtinpt
projection geographic
units ds
spheroid wgs84
datum wgs84
parameters

build xxtinpt line

/* converting the extents of the playbox into decimal seconds
/* with a 10 decimal second buffer

&s xmax2 [calc [calc %xmax% * 3600] + 10]
&s ymax2 [calc [calc %ymax% * 3600] + 10]
&s xmin2 [calc [calc %xmin% * 3600] - 10]
&s ymin2 [calc [calc %ymin% * 3600] - 10]

createtin xhytin .001 # # %xmin2% %ymin2% %xmax2% %ymax2% 0
  cover xhyds line hylnval 1
  &if %oceanpoly% &then cover xoceanx poly 0 5

  /* incorporate points from the traf dted cells to increase accuracy
  &s dtedcell [open xgddted.txt os -read]
  &do loop = 1 &to [calc %goodtot% - 1]
    &s cellname [read %dtedcell% rs]
    &s cellname [substr %cellname% 1 8]
    cover %cellname%3pt point spot 1
  &end
  &s junk [close %dtedcell%]

  /* Incorporate the four corner points of the tin area
  /* For some reason, this isn't working outside of the current hull!
  cover xxtinpt line -9999 2
end

kill xxtinpt all

/* Now deleting these 300 decimal second files to reduce clutter
&s dtedcell [open xgddted.txt os -read]
&do loop = 1 &to [calc %goodtot% - 1]
  &s cellname [read %dtedcell% rs]
  &s cellname [substr %cellname% 1 8]
  kill %cellname%3pt all
  kill %cellname%_3c all
&end
&s junk [close %dtedcell%]

/* Part of this thinning process will be done in a loop, by DTED
/* cell. Otherwise total file sizes would get out of hand.... you
/* could need over a gig to do a 100 degree cell playbox. At the
/* end of the loop we will have a bunch of text files with the
/* x & y coordinates, elevations, and idpolys.

&if [exists xtemppts -cover] &then kill xtemppts all
&if [exists xidentity -cover] &then kill xidentity all
&if [exists xxgrid -cover] &then kill xxgrid all

```

```

&if [exists xxtempgrd -grid] &then kill xxtempgrd all
&sys rm xxtin*.txt
&sys rm xxidentity.txt
&s counter 1

&do loop1 = %xmin% &to %xmax%
  &do loop2 = %ymin% &to %ymax%
    &if [%loop1% >= 0 && do /* if in the eastern hemisphere
      &if [length %loop1%] = 1 && do x 00%loop1%
        &else &if [length %loop1%] = 2 && do x 0%loop1%
        &else &s x %loop1%
        &s x e%x%
      &end
    &else &do /* if in the western hemisphere
      &s temp [abs %loop1%]
      &if [length %temp%] = 1 && do s x 00%temp%
        &else &if [length %temp%] = 2 && do s x 0%temp%
        &else &s x %temp%
        &s x w%x%
      &end
    &if [%loop2% >= 0 && do /* if in the northern hemisphere
      &if [length %loop2%] = 1 && do s y 0%loop2%
        &else &s y %loop2%
        &s y n%y%
      &end
    &else &do /* if in the southern hemisphere
      &s temp [abs %loop2%]
      &if [length %temp%] = 1 && do s y 0%temp%
        &else &s y %temp%
        &s y s%y%
      &end
    &s xmaxds [calc [%loop1% + 1] * 3600]
    &s ymaxds [calc [%loop2% + 1] * 3600]
    &s xminds [calc %loop1% * 3600]
    &s yminds [calc %loop2% * 3600]

    /* Creating resampled grids at 15 decimal seconds for use
    /* by the TIN creation program
    &if ^ [exists x%x%y%_2 -grid] &then &do
      &if ^ [exists x%x%y%grid -grid] &then &do
        tinlattice xhytin x%x%y%_2
        [calc %xminds% + 15], [calc %yminds% + 15]
        %xmaxds%, %ymaxds%
        ~
        15
      &end
    &else &do
      laticeresample x%x%y%grid x%x%y%_2
      [calc %xminds% + 15], [calc %yminds% + 15]
      ~
      ~
      15
    &end
  &end
/* Moving creation of the trafficability grid down here

```

```

  &if ^ [exists x%x%y%grid -grid] &then &do /* Fix comments here
later
  &if ^ [exists x%x%y%_15 -grid] &then &do
    tinlattice xhytin x%x%y%_15
    %xminds%, %yminds%
    %xmaxds%, %ymaxds%
    ~
    15
  &end
  &end

  /* Creating a 144 x 144 decimal seconds grid covering this
  /* cell's extents

  /* converting the extents of the playbox into decimal seconds
  &s xmin2 [calc %loop1% * 3600]
  &s ymin2 [calc %loop2% * 3600]
  &s xmax2 [calc %xmin2% + 3600]
  &s ymax2 [calc %ymin2% + 3600]

  /* Creating the cover with the grid on it called xxgrid
  create xxgrid

  projectdefine cover xxgrid
    projection geographic
    units ds
    spheroid wgs84
    datum wgs84
    parameters

  tables
    sel xxgrid.tic
    add
    1
    %xmin2%
    %ymin2%
    2
    %xmin2%
    %ymin2%
    3
    %xmax2%
    %ymax2%
    4
    %xmax2%
    %ymin2%
    ~
    quit

  ae
    coordinates keyboard
    ec xxgrid
    ef arc
    add
    &do loopx = %xmin2% &to %xmax2% &by 144
      2,%loopx%,%ymin2%
      2,%loopx%,%ymax2%
    &end

```

```

&do loopy = %ymin2% &to %ymax2% &by 144
  2,%xmin2%,%loopy%
  2,%xmax2%,%loopy%
&end
~
save
quit

tolerance xxgrid nodesnap 10
tolerance xxgrid weed 10
clean xxgrid
tables
  sel xxgrid.pat
  calc xxgrid-id = xxgrid#
quit

&if [exists x%x%%y%_3 -grid] &then kill x%x%%y%_3 all

/* Need to go into the grid and change all NODATA values to 0
grid
  xxtempgrd = con(isnull(x%x%%y%_2) == 1, 0, x%x%%y%_2)
  kill x%x%%y%_2 all
  rename xxtempgrd x%x%%y%_3
quit

gridpoint x%x%%y%_3 xtemppts spot
kill x%x%%y%_3 all
build xtemppts point
&if [exists xidentity -cover] &then kill xidentity all
identity xtemppts xxgrid xidentity point
kill xtemppts all
kill xxgrid all
build xidentity point
addxy xidentity

tables
  sel xidentity.pat
  sort xxgrid-id spot
  unload xxidentity.txt xxgrid-id spot x-coord y-coord ~
  COLUMNAR junk.txt

  /* Using a Perl script to sort the unloaded file by elevation,
  /* outputting the high and low of each cell to the final generate
-
  /* format file. The homogenous cells will be output to a
different
  /* text file and dealt with differently.
&sys .../amls/154tins.perl
  sort xidentity#
quit
&end
&end

/* Adding the requisite END to each file
&s fend END
&s out1 [open xxtin1.txt os -append]
&s junk [write %out1% %fend%]

&s junk [close %out1%]
&s out1 [open xxtin2.txt os -append]
&s junk [write %out1% %fend%]
&s junk [close %out1%]

&type [DATE -vfull]

/* Put together the individual cells for the trafficability aml
&if [exists xprvmerge -grid] &then kill xprvmerge all

&do loop1 = %xmin% &to %xmax%
  &do loop2 = %ymin% &to %ymax%
    &if %loop1% >= 0 &then &do      /* if in the eastern hemisphere
      &if [length %loop1%] = 1 &then &s x 0%loop1%
        &else &if [length %loop1%] = 2 &then &s x 0%loop1%
        &else &s x %loop1%
        &s x e%x%
      &end
    &else &do      /* if in the western hemisphere
      &s temp [abs %loop1%]
      &if [length %temp%] = 1 &then &s x 0%temp%
        &else &if [length %temp%] = 2 &then &s x 0%temp%
        &else &s x %temp%
        &s x w%x%
      &end
    &if %loop2% >= 0 &then &do      /* if in the northern hemisphere
      &if [length %loop2%] = 1 &then &s y 0%loop2%
        &else &s y %loop2%
        &s y n%y%
      &end
    &else &do      /* if in the southern hemisphere
      &s temp [abs %loop2%]
      &if [length %temp%] = 1 &then &s y 0%temp%
        &else &s y %temp%
        &s y s%y%
      &end
    &if [exists xprvmerge -grid] &then &do
      kill xprvmerge all
      rename xtempgrid xprvmerge
      grid
        xtempgrid = merge (xprvmerge, x%x%%y%_15)
      quit
      kill x%x%%y%_15 all
    &end
  &else &do
    copy x%x%%y%_15 xtempgrid
    rename x%x%%y%_15 xprvmerge
  &end
  &end
  &end

/* This grid, xxgrid15, will be used in the trafficability AML
kill xprvmerge all
rename xtempgrid xxgrid15

/* Now, back to our regular TIN...
/* Creating a second point coverage with the centers of the

```

```

/* homogenous cells, then creating a TIN with these two main
/* groups of points. Only doing this part if there are actually
/* cells in the grid that are homogenous.

/* Testing for the existence of homogenous cells.
&s tin2 [open xxtin2.txt os -read]
&s input [read %tin2% rs]
&if %input% = END &then &s tin2test .FALSE.
&else &s tin2test .TRUE.
&s junk [close %tin2%]

&if %tin2test% &then &do

/* First, recreate the 144DS grid covering the entire playbox area.
/* converting the extents of the playbox into decimal seconds
&s xmin2 [calc %xmin% * 3600]
&s ymin2 [calc %ymin% * 3600]
&s xmax2 [calc [calc %xmax% + 1] * 3600]
&s ymax2 [calc [calc %ymax% + 1] * 3600]

/* Creating the cover with the grid on it called xxgrid
create xxgrid

projectdefine cover xxgrid
  projection geographic
  units ds
  spheroid wgs84
  datum wgs84
  parameters

tables
  sel xxgrid.tic
  add
  1
  %xmin2%
  %ymin2%
  2
  %xmin2%
  %ymax2%
  3
  %xmax2%
  %ymax2%
  4
  %xmax2%
  %ymin2%
  ~
quit

ae
  coordinates keyboard
  ec xxgrid
  ef arc
  add
  &do loopx = %xmin2% &to %xmax2% &by 144
    2,%loopx%,%ymin2%
    2,%loopx%,%ymax2%
  &end

&do loopy = %ymin2% &to %ymax2% &by 144
  2,%xmin2%,%loopy%
  2,%xmax2%,%loopy%
&end
~
save
quit

tolerance xxgrid nodesnap 10
tolerance xxgrid weed 10
clean xxgrid
tables
  sel xxgrid.pat
  calc xxgrid-id = xxgrid#
quit

/* Second, we need to somehow pick out those cells which are
/* homogenous. All we have right now is a file full of point
/* values. Create a point coverage and then do an identity
/* with the big grid. This will take awhile but it only has
/* to be done once.

generate xtemppts
  input xxtin2.txt
  points
quit
build xtemppts point

tables
  additem xtemppts.pat spot 10 10 I
  &if [exists xxrelate.info -info] &then kill xxrelate.info
  define xxrelate.info
    pointid
    10
    10
    I
    spot
    10
    10
    I
  ~
  sel xxrelate.info
  add pointid spot FROM xxtin3.txt
  relate add
    elevs
    xxrelate.info
    info
    xtemppts-id
    pointid
    linear
    rw
  ~

  sel xtemppts.pat
  calc spot = elevs//spot
  relate drop
    elevs

```

```

~  

/* kill xxrelate.info  

quit  

build xtemppts point  

  
/* We have a point coverage with Z values, with a point for  

/* each square in the grid that is homogenous. Assign poly id  

/* values to this point coverage with an identity.  

  
&if [exists xidentity2 -cover] &then kill xidentity2 all  

identity xtemppts xxgrid xidentity2 point  

kill xtemppts all  

build xidentity2 point  

  
/* Set up a relate to flag polygons in the grid coverage.  

/* Relating xidentity.pat and xxgrid.pat  

/* We will keep the ones that are homogenous.  

tables  

  additem xxgrid.pat spot 10 10 i  

  relate add  

    hgpoly  

    xidentity2.pat  

    info  

    xxgrid-id  

    xxgrid-id  

    linear  

    rw  

~  

sel xxgrid.pat  

calc spot = -32767  

calc spot = hgpoly//spot  

relate drop  

  hgpoly  

~  

quit  

  
/* Write out the polygon centers to a point coverage, xxgridl.  

&if [exists xxgridl -cover] &then kill xxgridl all  

&if [exists xxgridpt -cover] &then kill xxgridpt all  

  
ae  

  ec xxgrid  

  build  

  ef point  

  sel all  

  put xxgridl  

  save  

quit  

build xxgridl point  

  
/* Use an identity again to relate the poly ids to the poly  

/* centers.  

identity xxgridl xxgrid xxgridpt point  

kill xxgridl all  

  
/* Assign elevations to this point coverage.  

  
tables  

  relate add  

  endelev  

  xxgrid.pat  

  info  

  xxgrid-id  

  xxgrid-id  

  linear  

  rw  

~  

sel xxgridpt.pat  

calc spot = endelev//spot  

relate drop  

  endelev  

~  

quit  

  
/* Delete the points with elevations of -32767; these are  

/* the polygons that are not homogenous.  

ae  

  ec xxgridpt  

  ef point  

  sel spot = -32767  

  &if [show number selected] ne 0 &then delete  

  save  

quit  

build xxgridpt point  

&end  

  
/* Find the extents one more time for use with the final TIN.  

&describe xtiles  

&s xmin %dsc%xmin%  

&if %xmin% ne [truncate %xmin%] &then &s xmin [truncate %xmin%]  

&s xmax %dsc%xmax%  

&if %xmax% ne [truncate %xmax%] &then &s xmax [calc [truncate %xmax%] +  

1]  

&s ymin %dsc$ymin%  

&if %ymin% ne [truncate %ymin%] &then &s ymin [truncate %ymin%]  

&s ymax %dsc$ymax%  

&if %ymax% ne [truncate %ymax%] &then &s ymax [calc [truncate %ymax%] +  

1]  

&s xmax %xmax% - 1  

&s ymax %ymax% - 1  

&s xmin2 [calc %xmin% * 3600]  

&s ymin2 [calc %ymin% * 3600]  

&s xmax2 [calc [calc %xmax% + 1] * 3600]  

&s ymax2 [calc [calc %ymax% + 1] * 3600]  

  
/* Testing for the existence of non-homogenous cells.  

&s tin1 [open xxtin1.txt os -read]  

&s input [read %tin1% rs]  

&if %input% = END &then &s tintest .FALSE.  

&else &s tintest .TRUE.  

&s junk [close %tin1%]  

  
/* Build a TIN, the final CBS TIN!!!, with the points from  

/* the file created by the perl script, xxtin1.txt, and the

```

```

/* point coverage just created.
&if [exists xxcbstin -tin] &then kill xxcbstin all
createtin xxcbstin # # %xmin2% %ymin2% %xmax2% %ymax2% 9999
  &if %tin2test% &then cover xxgridpt point spot 1
  &if %tin1test% &then generate xxtin1.txt point 1
end

&sys rm xxungentin.txt
ungeneratetin xxcbstin xxungentin.txt net
&sys mv xxungentin.txt.net xxungentin.txt

*****
/* Now run a perl script to reformat the text file and change to
/* units of 3 decimal seconds as needed by CBS. It works on the
/* ungenerated tin called xxungentin.txt
*****
&sys ..../ams/154tins2.perl

&do cov &list xidentity xidentity2
  &if [exists %cov% -cover] &then kill %cov% all
&end

/* Now I am launching the trafficability program.
&r 154traf

&ty Move xxfinal_tin.txt to the output directory as
&ty (name)_tin_low.trn!!!

&echo &off
display 9999
&ty [date -VFULL]
&watch &off
&return
*****
/* bail routine
*****
&routine bail
&severity &error &ignore
&severity &warning &ignore
&type An error has occurred in 154tins.aml
&type Bailing out of 154tins.aml
&s junk [close -ALL]
&return; &return &error

```

## B.10.2 154tins.perl

```

#! /usr/local/bin/perl

# -----
#   Program: 154tins.perl
#   Purpose: to reorder and thin a text file containing the DTED
#   points for the playbox.
# -----
#   Called By: 154tins.aml
#   Calls Made: none
# -----
#   Arguments: none
#   Globals: none
#   Locals:
# -----
#   Inputs: xxiden#.txt, the files with the point information
#   Outputs: xxtin1.txt, a GENERATE file to be used to create
#   a TIN. xxtin2.txt, an ID, X, Y file used to get homogenous
#   cell points into Arc. xxtin3.txt, an ID, Z file used to
#   get elevations values into ARC for homogenous cells.
# -----
#   History: 10 October 1997 - Tobi Sellekaerts
#           steinbet@email.grafenwoehr.army.mil
#           Terrain Simulations - 7th ATC - Grafenwoehr, Germany
# -----

open(INPUT, "xxidentity.txt");
open(OUT1, ">>xxtin1.txt");      # this will be the GENERATE file
open(OUT2, ">>xxtin2.txt");      # used to get x,y values into ARC
open(OUT3, ">>xxtin3.txt");      # used to get elev values into ARC

# z signifies the elevation value.
# id is the polygon of the grid that was superimposed over the
# point coverage. Many different points will have the same id.
# hi and lo versions of z and id2 are self explanatory.
$counter = 1; # so there are sequential numbers in the xxtin1.txt
$counter2 = 1; # so there are sequential numbers in xxtin2 & xxtin3

while($in = <INPUT>) {
    $id = substr($in, 0, 5) * 1;
    if($id == $previd) {
        # we are still working on the current polygon
        # find z-values and compare to hi and low
        $z = substr($in, 5, 12) * 1;
        if($z > $hiz) {
            $hiz = $z;
            $shix = substr($in, 17, 18) * 1;
            $shiy = substr($in, 35, 18) * 1;
        } elsif($z < $loz) {
            $loz = $z;
            $lox = substr($in, 17, 18) * 1;
            $loy = substr($in, 35, 18) * 1;
        }
    } else {
        # we are at the end of the list of points for this poly

```

```

# write out the final information to the output file
if($previd > 0) {      # only if this isn't point #1
    if($hiz == $loz) {
        # if the two elevations are the same, this is a
        # homogenous cell
        print OUT3 "$previd, $hiz\n";
        print OUT2 "$previd, $hix, $hiy\n";
    } else {
        # this is not a homogenous cell
        print OUT1 "$counter, $lox, $loy, $loz\n";
        ++$counter;
        print OUT1 "$counter, $hix, $hiy, $hiz\n";
        ++$counter;
    }
}

# read in the info from this new point to begin a
# new polygon
$previd = substr($in, 0, 5) * 1;
$sz = substr($in, 5, 12) * 1;
$hiz = $sz;
$loz = $hiz;
$hix = substr($in, 17, 18) * 1;
$hiy = substr($in, 35, 18) * 1;
$lox = substr($in, 17, 18) * 1;
$loy = substr($in, 35, 18) * 1;
}
}

# working on the last point read in
# we are at the end of the list of points for this poly
# write out the final information to the output file
if($hiz == $loz) {
    # if the two elevations are the same, this is a
    # homogenous cell
    print OUT3 "$id, $hiz\n";
    print OUT2 "$id, $hix, $hiy\n";
} else {
    # if the two elevations are not the same, this is
    # not a homogenous cell
    print OUT1 "$counter, $lox, $loy, $loz\n";
    ++$counter;
    print OUT1 "$counter, $hix, $hiy, $hiz\n";
    ++$counter;
}

close(INPUT);
close(OUT1);
close(OUT2);
close(OUT3);

```

### B.10.3 154tins2.perl

```

#!/usr/local/bin/perl

# -----
#     Program: 154tins2.perl
#     Purpose: to reformat and change the units of an ungereratetin
#     export file created by Arc/Info and make it into the
#     appropriate CBS 1.5.4 file format.
# -----
#     Called By: 154tins.aml
#     Calls Made: none
#
# -----
#     Arguments: none
#     Globals: none
#
# -----
#     Inputs: the ungereratetin file, xxungentin.txt
#     Outputs: a properly formatted CBS file, xxfinal_tin.txt
# -----
#     History: date - 18 August 1997 - Tobi Sellekaerts
#             steinbet@email.grafenwoehr.army.mil
#             http://199.56.131.202/~tobi/
#             Terrain Simulations - 7th ATC - Grafenwoehr, Germany
# -----
# first, extract each section of the ungereratetin file into a file
# with that feature only
print "Extracting features into their own files...\n";
open(IN, ">xxungentin.txt");
open(PTS, ">xxpts_tin.txt"); # points
open(LNS, ">xxlns_tin.txt"); # lines
open(PLY, ">xxply_tin.txt"); # triangles or polygons

# do the loop for points
$a = 1;
$in = <IN>;      # strip out the line that says 'NODES'
while($a) {
    $in = <IN>;
    chop($in);
    $test = substr($in, 0, 5);
    if($test eq "EDGES") {
        $a = 0;
    } else {
        print PTS "$in\n";
    }
}

# do the loop for lines
$a = 1;
while($a) {
    $in = <IN>;
    chop($in);
    $test = substr($in, 0, 5);
    if($test eq "TRIAN") {
        $a = 0;
    }
}

```

```

} else {
    print LNS "$in\n";
}
}

# do the loop for polygons
$a = 1;
while($in = <IN>) {
    chop($in);
    print PLY "$in\n";
}

close(IN);
close(PTS);
close(LNS);
close(PLY);

# change the units in the point file now, and change format
print "Changing the units in the point file...\n";
open(IN, "xxpts_tin.txt");
open(OUT, ">xxpts2_tin.txt");

while($in = <IN>) {
    $num = substr($in, 0, 12) * 1;
    $x = substr($in, 12, 20);
    $y = substr($in, 32, 24);
    $elev = substr($in, 56, 15) * 1;
    $x = int $x;
    $y = int $y;
    $elev = int $elev;
    $x *= 3;
    $y *= 3;
    printf OUT "%12u %12u %12u %12u\n", $num, $x, $y, $elev;
}

close(IN);
close(OUT);

system("date");

# create a temporary file with the adjacent polygon id's
print "Creating a temporary file with the adjacent polygon id's...\n";
system("cp xxply_tin.txt xxply2_tin.txt"); # need a copy to work with
also
open(IN, "xxply_tin.txt");
open(OUT, ">xxadj_tin.txt");

while($ply = <IN>) {
    $plyct = substr($ply, 0, 12) * 1;
    $seg1 = substr($ply, 12, 12) * 1;
    $seg2 = substr($ply, 24, 12) * 1;
    $seg3 = substr($ply, 36, 12) * 1;

    # test for each segment in turn
    # segment #1
    open(TEST, "xxply2_tin.txt");
    $adj1 = 0;
}

```

```

while($test = <TEST>) {
    $testct = substr($test, 0, 12) * 1;
    if($testct != $plyct) {
        $test1 = substr($test, 12, 12) * 1;
        $test2 = substr($test, 24, 12) * 1;
        $test3 = substr($test, 36, 12) * 1;
        if($test1 == $seg1) {
            $adj1 = $testct;
        } elsif($test2 == $seg1) {
            $adj1 = $testct;
        } elsif($test3 == $seg1) {
            $adj1 = $testct;
        }
    }
}
close(TEST);

# segment #2
open(TEST, "xxply2_tin.txt");
$adj2 = 0;
while($test = <TEST>) {
    $testct = substr($test, 0, 12) * 1;
    if($testct != $plyct) {
        $test1 = substr($test, 12, 12) * 1;
        $test2 = substr($test, 24, 12) * 1;
        $test3 = substr($test, 36, 12) * 1;
        if($test1 == $seg2) {
            $adj2 = $testct;
        } elsif($test2 == $seg2) {
            $adj2 = $testct;
        } elsif($test3 == $seg2) {
            $adj2 = $testct;
        }
    }
}
close(TEST);

# segment #3
open(TEST, "xxply2_tin.txt");
$adj3 = 0;
while($test = <TEST>) {
    $testct = substr($test, 0, 12) * 1;
    if($testct != $plyct) {
        $test1 = substr($test, 12, 12) * 1;
        $test2 = substr($test, 24, 12) * 1;
        $test3 = substr($test, 36, 12) * 1;
        if($test1 == $seg3) {
            $adj3 = $testct;
        } elsif($test2 == $seg3) {
            $adj3 = $testct;
        } elsif($test3 == $seg3) {
            $adj3 = $testct;
        }
    }
}
close(TEST);

```

```

# printf "%12u %11u %11u %11u\n", $plyct, $adj1, $adj2, $adj3;
printf OUT "%12u %11u %11u %11u\n", $plyct, $adj1, $adj2, $adj3;
}

close(IN);
close(OUT);

system("date");

# now combine all this information and make the final output file
print "Combining all this information and making the final output
file...\n";
open(PLY, "xxply_tin.txt");
open(OUT, ">xxfinal_tin.txt");
open(ADJ, "xxadj_tin.txt");

print OUT "600 0 1 0\n"; # necessary header line
$counter = 1;
$counter *= 1;
while($ply = <PLY>) {
    $seg1 = substr($ply, 12, 12) * 1;
    $seg2 = substr($ply, 24, 12) * 1;
    $seg3 = substr($ply, 36, 12) * 1;

    # find each segment in the lines file
    open(LNS, "xxlns_tin.txt");
    while($lns = <LNS>) {
        $lnsno = substr($lns, 0, 12) * 1;
        $pt1 = substr($lns, 12, 12) * 1;
        $pt2 = substr($lns, 24, 12) * 1;
        if($lnsno == $seg1) {
            $seg1pt1 = $pt1;
            $seg1pt2 = $pt2;
        } elsif($lnsno == $seg2) {
            $seg2pt1 = $pt1;
            $seg2pt2 = $pt2;
        } elsif($lnsno == $seg3) {
            $seg3pt1 = $pt1;
            $seg3pt2 = $pt2;
        }
    }
    close(LNS);

    # testing to see if all coordinates are in the correct order
    # $test will be false when everything is correct
    # this problem should be used as a test in a programming class!!!
    $test = 1;
    while($test) {
        if($seg1pt2 != $seg2pt1) {
            $test += 1;
            $temp = $seg1pt1;
            $seg1pt1 = $seg1pt2;
            $seg1pt2 = $temp;
        }
        if($seg1pt2 != $seg2pt1) {
            $temp = $seg2pt1;
            $seg2pt1 = $seg2pt2;
            $seg2pt2 = $temp;
        }
    }
}

```

```

        }
    } elsif($seg2pt2 != $seg3pt1) {
        $test += 1;
        $temp = $seg2pt1;
        $seg2pt1 = $seg2pt2;
        $seg2pt2 = $temp;
        if($seg2pt2 != $seg3pt1) {
            $temp = $seg3pt1;
            $seg3pt1 = $seg3pt2;
            $seg3pt2 = $temp;
        }
    } elsif($seg3pt2 != $seg1pt1) {
        $test += 1;
        $temp = $seg3pt1;
        $seg3pt1 = $seg3pt2;
        $seg3pt2 = $temp;
    } else {
        $test = 0;
    }
}

# find each point in the point file
open(PTS, "xxpts2_tin.txt");
while($pts = <PTS>) {
    $ptno = substr($pts, 0, 12) * 1;
    $ptx = substr($pts, 12, 13) * 1;
    $pty = substr($pts, 25, 13) * 1;
    $elev = substr($pts, 38, 13) * 1;
    if($ptno == $seg1pt1) {
        $seg1x = $ptx;
        $seg1y = $pty;
        $seg1elev = $elev;
    } elsif($ptno == $seg2pt1) {
        $seg2x = $ptx;
        $seg2y = $pty;
        $seg2elev = $elev;
    } elsif($ptno == $seg3pt1) {
        $seg3x = $ptx;
        $seg3y = $pty;
        $seg3elev = $elev;
    }
}
close(PTS);

# of course simply printing out the coordinates is much too easy...
# calculate necessary differences
$delta1x = $seg1x - $seg2x;
$delta1y = $seg1y - $seg2y;
$delta1x = $seg1x - $seg3x;
$delta1y = $seg1y - $seg3y;

# extract the adjacent triangle id's
$adj = <ADJ>;
$adj1 = substr($adj, 12, 12) * 1;
$adj2 = substr($adj, 24, 12) * 1;
$adj3 = substr($adj, 36, 12) * 1;

```

```

# now write out the information
# see CBS 1.5.4 file folder for more format information
print OUT "601 3 $counter 3 $adj1 $adj2 $adj3\n";
print OUT "$seg1x $segly $seg1elev\n";
print OUT "$delta12x $delta12y $seg2elev\n";
print OUT "$delta13x $delta13y $seg3elev\n";

$counter += 1;
}

close(ADJ);
close(OUT);
close(PLY);

system("rm xxpts_tin.txt");
system("rm xxpts2_tin.txt");
system("rm xxlns_tin.txt");
system("rm xxply_tin.txt");
system("rm xxply2_tin.txt");
system("rm xxadj_tin.txt");
system("rm xxungentin.txt");
system("date");

```

## B.11 Bridges

### B.11.1 154bridges.aml

```

/*154bridges.aml
/*
-----*
*      Program: 154bridges.aml
*      Purpose: to create the bridges output file for CBS 1.5.4
*      What I'm going to do is recreate the bridges coverage using
*      as input the text files I already created for trans and
*      hydro. Yes, going back a step to go forward two. But this
*      is the easiest way to ensure that I use CBS road & river ids.
*      -----
*      Called By: dcwcbs154.aml
*      Calls Made: 154bridges1.perl, 154bridges2.perl,
*                  154bridges3.perl, 154bridges4.perl
*      -----
*      Arguments: none
*      Globals: .name, .dirname
*      Locals:
*      -----
*      Inputs: the text files of roads & hydro for CBS
*      Outputs: a properly formatted bridges file,
*              %.name%_bridges.txt
*      -----
*      History: 19 August 1997 - Tobi Sellekaerts
*              steinbet@email.grafenwoehr.army.mil
*              http://199.56.131.202/~tobi/
*              Terrain Simulations - 7th ATC - Grafenwoehr, Germany
*      -----
&severity &error &routine bail
&ty [date -VFULL]
&echo &on
/* Checking on the existence of coverage names needed by the program
&do cov &list xxbridges
    &if [exists %cov% -cover] &then kill %cov% all
&end

&sys rm xxnodearc?.txt
&sys rm xxnodexy.txt

/* First, call a perl script to reformat the roads and rivers files
/* so they can be used by generate
&sys ../amls/154bridges1.perl

/* create the bridges file and set the tolerances
create xxbridges
tolerance xxbridges nodesnap .000005
tolerance xxbridges weed .000005

/* generate the bridges coverage from generate.txt files
/* this file will include all the roads and rivers
generate xxbridges
    input xxrd.txt
    lines
    input xxhy.txt

```

```

lines
quit

build xxbridges line
build xxbridges node
addxy xxbridges node

/* Add another item to represent the id so if I have to do anything
/* to the coverage, the ids won't get reassigned.
additem xxbridges.aat xxbridges.aat cbsid 10 10 i
additem xxbridges.aat xxbridges.aat from_n 10 10 i
additem xxbridges.aat xxbridges.aat to_n 10 10 i
tables
sel xxbridges.aat
calc cbsid = xxbridges-id
calc from_n = fnode#
calc to_n = tnode#

/* Unloading arc information into text files to be used by perl
sort from_n
unload xxnodearc1.txt from_n cbsid columnar junk.txt
sort to_n
unload xxnodearc2.txt to_n cbsid columnar junk.txt
sort xxbridges#

/* Unloading node x-y coordinates for perl script
sel xxbridges.nat
unload xxnodexy.txt xxbridges-id x-coord y-coord columnar junk.txt
quit

&sys ../amls/154bridges2.perl

/* Back into aml for some more processing... taking advantage of the fast
/* sort in tables
tables
&if [exists int_relate.nat -info] &then kill int_relate.nat
define int_relate.nat
  cbsid1; 9; i
  from_to; 1; 2; i
  cbsid2; 9; i
~
sel int_relate.nat
add cbsid1 from_to cbsid2 FROM xxmult2.txt
sort cbsid1
unload xxunload.txt cbsid1 from_to cbsid2 columnar junk.txt
kill int_relate.nat
quit

/* Now back to perl to add the necessary text to the output road and
/* river files
&sys ../amls/154bridges3.perl

*****
/* Now officially working on bridges... finally!
***** 

/* Doing some more processing for the next perl script - sorting again

```

```

tables
&if [exists sort_relate.nat -info] &then kill sort_relate.nat
define sort_relate.nat
  nodeid; 9; i
  cbsid1; 9; i
  type1; 1; 2; i
  cbsid2; 9; i
  type2; 1; 2; i
  nodex; 10; 10; n; 6
  nodey; 10; 10; n; 6
  ~
sel sort_relate.nat
add nodeid cbsid1 type1 cbsid2 type2 nodex nodey FROM xxbridges2.txt
sort cbsid1
unload xxbridges3.txt nodeid cbsid1 type1 cbsid2 type2 nodex nodey
columnar junk.txt
quit

/* finding angles
/* for each of the nodes that is a bridge.
&s inline [open xxbridges2.txt os -read]
&s out [open xxangle.txt os -write]
&s in [read %inline% rs]
&s tester .FALSE.
&if %rs% ne 102 &then &s tester .TRUE.

ae
ec xxbridges
ef arc
&do &while %tester%
  &s in [unquote %in%]
  &s nodeid [extract 1 %in%]
  &s roadid [extract 2 %in%]
  &s x [extract 6 %in%]
  &s y [extract 7 %in%]
  sel cbsid = %roadid%
  &s angle [show arc [show select 1] orientation %x% %y%]
  &s junk [write %out% %nodeid%]
  &s junk [write %out% %angle%]
  &s in [read %inline% rs]
  &s tester .FALSE.
  &if %rs% ne 102 &then &s tester .TRUE.
&end
quit

&s junk [close %out%]
&s junk [close %inline%]
&sys ../amls/154bridges4.perl

/* Moving around and rename various files
&sys cp final_bridges.txt %.dirname%/.name%_bridgs.txt
&if [exists %.dirname%/.name%_bridgs.txt -file] &then
  &sys rm final_bridges.txt

&sys cp xxhydroout1.txt %.dirname%/.name%_strms1.txt
&sys cp xxhydroout2.txt %.dirname%/.name%_strms2.txt
&sys cp xxhydroout3.txt %.dirname%/.name%_strms3.txt

```

```

&if [exists %.dirname%/.name%_strms3.txt -file] &then
  &sys rm xxhydroout?.txt

&sys cp xxtransout1.txt %.dirname%/.name%_roads1.txt
&sys cp xxtransout2.txt %.dirname%/.name%_roads2.txt
&sys cp xxtransout3.txt %.dirname%/.name%_roads3.txt
&if [exists %.dirname%/.name%_roads3.txt -file] &then
  &sys rm xx*.txt

&ty [date -VFULL]
&return
/*****
/* bail routine
/*****
&routine bail
&echo &off
&severity &error &ignore
&severity &warning &ignore
&type An error has occurred in 154bridges.aml
&type Bailing out of 154bridges.aml
&return; &return &error

```

## B.11.2 154bridges1.perl

```

#! /usr/local/bin/perl

# 154bridges1.perl
# -----
#   Program: 154bridges1.perl
#   Purpose: to reformat the hydro and trans CBS files so they
#   can be used by generate in ARC
# -----
#   Called By: 154bridges.aml
#   Calls Made: none
# -----
#   Arguments: none
#   Globals: none
#   Locals:
# -----
#   Inputs: hydro and trans CBS 1.5.4 files
#   Outputs: generate hydro and trans files
# -----
#   History: date - 19 August 1997 - Tobi Sellekaerts
#             steinbet@email.grafenwoehr.army.mil
#             http://199.56.131.202/~tobi/
#             Terrain Simulations - 7th ATC - Grafenwoehr, Germany
# -----



# now working with trans
open(OUT, ">xxrd.txt");
open(IN, "xxtransout1.txt");
while($in = <IN>) {
    if(substr($in, 0, 3) ne "END") { # don't execute at EOF
        if(substr($in, 0, 1) ne "!") { # don't execute for comments
            $sp1 = index($in, " ", 2);
            $sp2 = index($in, " ", 5);
            $length = $sp2 - $sp1;
            $verts = substr($in, $sp1, $length) * 1;
            $temp = 8;
            if($verts > 99) {
                $temp = 10;
            }
            $sp3 = index($in, " ", $temp);
            $sp4 = index($in, " ", ($temp + 2));
            $length = $sp4 - $sp3;
            $id = substr($in, $sp3, $length) * 1;
            print OUT "$id\n";
            $in = <IN>; # get rid of the second header line
            for ($i = 0; $i < $verts; ++$i) {
                $in = <IN>;
                $coordx = substr($in, 0, 10);
                if(substr($coordx, 0, 1) eq " ") {
                    $coordx = substr($coordx, 1)
                }
                if(substr($coordx, 0, 1) eq " ") {
                    $coordx = substr($coordx, 1)
                }
                $coordy = substr($in, 11, 11);
                if(substr($coordy, 0, 1) eq " ") {

```

```

        $coordy = substr($coordy, 1)
    }
    if(substr($coordy, 0, 1) eq " ") {
        $coordy = substr($coordy, 1)
    }
    if(substr($coordy, 0, 1) eq " ") {
        $coordy = substr($coordy, 1)
    }
    print OUT "$coordx, $coordy\n";
}
$in = <IN>;      # get rid of the line with the END
print OUT "END\n";
}
}
close(IN);
open(IN, "xxtransout2.txt");
while($in = <IN>) {
    if(substr($in, 0, 3) ne "END") {    # don't execute at EOF
        if(substr($in, 0, 1) ne "!") {    # don't execute for comments
            $sp1 = index($in, " ", 2);
            $sp2 = index($in, " ", 5);
            $length = $sp2 - $sp1;
            $verts = substr($in, $sp1, $length) * 1;
            $temp = 8;
            if($verts > 99) {
                $temp = 10;
            }
            $sp3 = index($in, " ", $temp);
            $sp4 = index($in, " ", ($temp + 2));
            $length = $sp4 - $sp3;
            $id = substr($in, $sp3, $length) * 1;
            print OUT "$id\n";
            $in = <IN>;      # get rid of the second header line
            for ($i = 0; $i < $verts; ++$i) {
                $in = <IN>;
                $coordx = substr($in, 0, 10);
                if(substr($coordx, 0, 1) eq " ") {
                    $coordx = substr($coordx, 1)
                }
                if(substr($coordx, 0, 1) eq " ") {
                    $coordx = substr($coordx, 1)
                }
                $coordy = substr($in, 11, 11);
                if(substr($coordy, 0, 1) eq " ") {
                    $coordy = substr($coordy, 1)
                }
                if(substr($coordy, 0, 1) eq " ") {
                    $coordy = substr($coordy, 1)
                }
                if(substr($coordy, 0, 1) eq " ") {
                    $coordy = substr($coordy, 1)
                }
                print OUT "$coordx, $coordy\n";
            }
            $in = <IN>;      # get rid of the line with the END
            print OUT "END\n";
        }
    }
}

```

```

}
}

close(IN);
open(IN, "xxtransout3.txt");
while($in = <IN>) {
    if(substr($in, 0, 3) ne "END") {      # don't execute at EOF
        if(substr($in, 0, 1) ne "!") {      # don't execute for comments
            $sp1 = index($in, " ", 2);
            $sp2 = index($in, " ", 5);
            $length = $sp2 - $sp1;
            $verts = substr($in, $sp1, $length) * 1;
            $temp = 8;
            if($verts > 99) {
                $temp = 10;
            }
            $sp3 = index($in, " ", $temp);
            $sp4 = index($in, " ", ($temp + 2));
            $length = $sp4 - $sp3;
            $id = substr($in, $sp3, $length) * 1;
            print OUT "$id\n";
            $in = <IN>;                      # get rid of the second header line
            for ($i = 0; $i < $verts; ++$i) {
                $in = <IN>;
                $coordx = substr($in, 0, 10);
                if(substr($coordx, 0, 1) eq " ") {
                    $coordx = substr($coordx, 1)
                }
                if(substr($coordx, 0, 1) eq " ") {
                    $coordx = substr($coordx, 1)
                }
                $coordy = substr($in, 11, 11);
                if(substr($coordy, 0, 1) eq " ") {
                    $coordy = substr($coordy, 1)
                }
                if(substr($coordy, 0, 1) eq " ") {
                    $coordy = substr($coordy, 1)
                }
                if(substr($coordy, 0, 1) eq " ") {
                    $coordy = substr($coordy, 1)
                }
                print OUT "$coordx, $coordy\n";
            }
            $in = <IN>;                      # get rid of the line with the END
            print OUT "END\n";
        }
    }
}
close(IN);
print OUT "END\n";
close(OUT);

# now working with hydro
open(OUT, ">xxhy.txt");
open(IN, "xxhydroout1.txt");
while($in = <IN>) {
    if(substr($in, 0, 3) ne "END") {      # don't execute at EOF

```

```

if(substr($in, 0, 1) ne "!=") { # don't execute for comments
    $sp1 = index($in, " ", 2);
    $sp2 = index($in, " ", 5);
    $length = $sp2 - $sp1;
    $verts = substr($in, $sp1, $length) * 1;
    $temp = 8;
    if($verts > 99) {
        $temp = 10;
    }
    $sp3 = index($in, " ", $temp);
    $sp4 = index($in, " ", ($temp + 2));
    $length = $sp4 - $sp3;
    $id = substr($in, $sp3, $length) * 1;
    print OUT "$id\n";
    $in = <IN>; # get rid of the second header line
    for ($i = 0; $i < $verts; ++$i) {
        $in = <IN>;
        $coordx = substr($in, 0, 10);
        if(substr($coordx, 0, 1) eq " ") {
            $coordx = substr($coordx, 1)
        }
        if(substr($coordx, 0, 1) eq " ") {
            $coordx = substr($coordx, 1)
        }
        $coordy = substr($in, 11, 11);
        if(substr($coordy, 0, 1) eq " ") {
            $coordy = substr($coordy, 1)
        }
        if(substr($coordy, 0, 1) eq " ") {
            $coordy = substr($coordy, 1)
        }
        if(substr($coordy, 0, 1) eq " ") {
            $coordy = substr($coordy, 1)
        }
        print OUT "$coordx, $coordy\n";
    }
    $in = <IN>; # get rid of the line with the END
    print OUT "END\n";
}
}

close(IN);
open(IN, "xxhydroout2.txt");
while($in = <IN>) {
    if(substr($in, 0, 3) ne "END") { # don't execute at EOF
        if(substr($in, 0, 1) ne "!=") { # don't execute for comments
            $sp1 = index($in, " ", 2);
            $sp2 = index($in, " ", 5);
            $length = $sp2 - $sp1;
            $verts = substr($in, $sp1, $length) * 1;
            $temp = 8;
            if($verts > 99) {
                $temp = 10;
            }
            $sp3 = index($in, " ", $temp);
            $sp4 = index($in, " ", ($temp + 2));
            $length = $sp4 - $sp3;

```

```

$id = substr($in, $sp3, $length) * 1;
print OUT "$id\n";
$in = <IN>; # get rid of the second header line
for ($i = 0; $i < $verts; ++$i) {
    $in = <IN>;
    $coordx = substr($in, 0, 10);
    if(substr($coordx, 0, 1) eq " ") {
        $coordx = substr($coordx, 1)
    }
    if(substr($coordx, 0, 1) eq " ") {
        $coordx = substr($coordx, 1)
    }
    $coordy = substr($in, 11, 11);
    if(substr($coordy, 0, 1) eq " ") {
        $coordy = substr($coordy, 1)
    }
    if(substr($coordy, 0, 1) eq " ") {
        $coordy = substr($coordy, 1)
    }
    print OUT "$coordx, $coordy\n";
}
$in = <IN>; # get rid of the line with the END
print OUT "END\n";
}
}
close(IN);
open(IN, "xxhydroout3.txt");
while($in = <IN>) {
    if(substr($in, 0, 3) ne "END") { # don't execute at EOF
        if(substr($in, 0, 1) ne "!=") { # don't execute for comments
            $sp1 = index($in, " ", 2);
            $sp2 = index($in, " ", 5);
            $length = $sp2 - $sp1;
            $verts = substr($in, $sp1, $length) * 1;
            $temp = 8;
            if($verts > 99) {
                $temp = 10;
            }
            $sp3 = index($in, " ", $temp);
            $sp4 = index($in, " ", ($temp + 2));
            $length = $sp4 - $sp3;
            $id = substr($in, $sp3, $length) * 1;
            print OUT "$id\n";
            $in = <IN>; # get rid of the second header line
            for ($i = 0; $i < $verts; ++$i) {
                $in = <IN>;
                $coordx = substr($in, 0, 10);
                if(substr($coordx, 0, 1) eq " ") {
                    $coordx = substr($coordx, 1)
                }
                if(substr($coordx, 0, 1) eq " ") {
                    $coordx = substr($coordx, 1)
                }

```

```

$coordy = substr($in, 11, 11);
if(substr($coordy, 0, 1) eq " ") {
    $coordy = substr($coordy, 1)
}
if(substr($coordy, 0, 1) eq " ") {
    $coordy = substr($coordy, 1)
}
if(substr($coordy, 0, 1) eq " ") {
    $coordy = substr($coordy, 1)
}
print OUT "$coordx, $coordy\n";
}
$in = <IN>;      # get rid of the line with the END
print OUT "END\n";
}
}
close(IN);
print OUT "END\n";
close(OUT);

```

### B.11.3 154bridges2.perl

```

#! /usr/local/bin/perl

# Warning - the logic of this program is awful. It evolved painfully,
# like a mutant gene that by darwinian logic should not have been
# allowed to survive. But it does the job.

# 154bridges2.perl
# -----
#     Program: 154bridges2.perl
#     Purpose: this program actually has two functions. It adds
#             intersection information to the river and road files, and
#             creates a bridges output file for CBS 1.5.4
# -----
#     Called By: 154bridges.aml
#     Calls Made: none
# -----
#     Arguments: none
#     Globals: none
#     Locals:
# -----
#     Inputs: xxnodearc1.txt - from_node and arc listing
#             xxnodearc2.txt - to_node and arc listing
#             xxhydstat.txt - statistics about where each hydro
#                             output file begins
#             xxrdrstat.txt - statistics about where each road
#                             output file begins
#     Outputs: xxbr_final.txt - formatted CBS bridges file
# -----
#     History: date - 21 August 1997 - Tobi Sellekaerts
#               steinbet@email.grafenwoehr.army.mil
#               http://199.56.131.202/~tobi/
#               Terrain Simulations - 7th ATC - Grafenwoehr, Germany
# -----
# =====
# =====
# first half of the program - add the intersection information to the
# road and river files
# =====
# =====

# read in xxnodearc1.txt and xxnodearc2.txt and combine them into a
# single file with the node #, from_arc, and to_arc on the same line

open(IN1, "xxnodearc1.txt"); # these are the from_arcs
open(IN2, "xxnodearc2.txt"); # these are the to_arcs
open(OUT, ">xxnodearc.txt"); # temp output file #1

# using the following variables to test for truth for the loop
# each will be true when each file runs out of lines
$a = 1;      # for input #1
$b = 1;      # for input #2

$in1 = <IN1>;
$in2 = <IN2>;

```

```

$node1 = substr($in1, 0, 10) * 1;
$node2 = substr($in2, 0, 10) * 1;
$id1 = substr($in1, 10, 10) * 1;
$id2 = substr($in2, 10, 10) * 1;

while($a && $b) {                                # print out whichever number is
less
    if($node1 <= $node2) {
        print OUT "1$in1";      # retaining 'from' information
        $in1 = <IN1>;          # when a line is printed, read in the
        $node1 = substr($in1, 0, 10) * 1; # next item from that list
        if($node1 == 0) {
            $a = 0;
        }
    } elsif($node2 <= $node1) {
        print OUT "2$in2";      # retaining 'to' information
        $in2 = <IN2>;
        $node2 = substr($in2, 0, 10) * 1;
        if($node2 == 0) {
            $b = 0;
        }
    }
}
}

close(IN1);
close(IN2);
close(OUT);

# next we are going to take out all the lines in the text file
# where a node number isn't repeated at least once
$temp = 0;
open(IN, "xxnodearc.txt");
open(OUT, ">xxmult.txt");
$in1 = <IN>;
while($in2 = <IN>) {
    $node1 = substr($in1, 1, 10) * 1;
    $id1 = substr($in1, 11, 10) * 1;
    $node2 = substr($in2, 1, 10) * 1;
    $id2 = substr($in2, 11, 10) * 1;
    if($node1 == $node2) {
        if($temp != $node2) {
            print OUT "$in1";
        }
        print OUT "$in2";
        $temp = $node2;
    }
    $in1 = $in2;
}
system("rm xxnodearc.txt");

# so now we have a text file that looks like this:
# 1      18      200106
# 1      18      200007
# 1      189     100286
# 1      189     100099
# 1      194     203474
# 1      194     203508

```

```

# showing the type of arc (1-from, 2-to), node#, and cbs id number
# for each arc.  cbsid >= 200000 is a river, < is a road.
# now we need to take this information and place it into another text
# file with one line for each road that has an entry.

# load the information from the statistics files into variables
open(STATS, "xxrdstat.txt");      # working on roads first
$in = <STATS>;                  # the first one will always be 100001
$in = <STATS>;                  # the id at the beginning of the second file
chop($in);
$rdbegin2 = $in * 1;
$in = <STATS>;                  # the id at the beginning of the third file
chop($in);
$rdbegin3 = $in * 1;
close(STATS);

open(STATS2, "xxhydstat.txt");    # working on hydro second
$in = <STATS2>;                # the first one will always be 200001
$in = <STATS2>;                # the id at the beginning of the second file
chop($in);
$hybegin2 = $in * 1;
$in = <STATS2>;                # the id at the beginning of the third file
chop($in);
$hybegin3 = $in * 1;
close(STATS2);

# using a truth variable to determine if we are on a node with more
# than two arcs coming out from it (will be true)
# if it is true, write the nodeid out to another file, xxmult2.txt
$gt2 = 0;

# creating another text file, xxbridges.txt, where I will write out
# the node id and road and river numbers if there is a bridge there
open(BRIDGE, ">xxbridges.txt");

# read in the first entry from the duplicate nodes file
open(IN, "xxmult.txt");
open(OUT, ">xxmult2.txt");       # the name of the output text file
$in = <IN>;
$cbsid1 = substr($in, 11, 10) * 1;
$nodeid1 = substr($in, 1, 10) * 1;
$direct1 = substr($in, 0, 1) * 1;
$count = 0;                      # will be total number of entries to the mult2.txt file
$brcount = 0;                     # the total number of entries to the bridges.txt file

# cycle through the duplicate nodes file
while($in = <IN>) {
    $cbsid2 = substr($in, 11, 10) * 1;
    $nodeid2 = substr($in, 1, 10) * 1;
    $direct2 = substr($in, 0, 1) * 1;

    # check to see that the nodeid values match.  if not, skip to the
    # next value.  This seems redundant, but necessary because they
    # won't always come in pairs

    if($nodeid1 == $nodeid2) {

```

```

# if the segments are different types (one a road and one
# a river) then output the node and segment ids to the
# bridges text file
if((int $cbsid1/100000) != (int $cbsid2/100000)) {
    if($cbsid1 > $cbsid2) {
        print BRIDGE "$nodeid1 $cbsid2 $direct2 $cbsid1
$cdirect1\n";
        ++$brcount;
    } else {
        print BRIDGE "$nodeid1 $cbsid1 $direct1 $cbsid2
$cdirect2\n";
        ++$brcount;
    }
}

if($gt2 != 1) {
    print OUT "$cbsid1 $direct1 $cbsid2\n";
    ++$count;
}
print OUT "$cbsid2 $direct2 $cbsid1\n";
++$count;
$gt2 = 1;
} else {
    $gt2 = 0;
}
$cbsid1 = $cbsid2;
$direct1 = $direct2;
$nodeid1 = $nodeid2;
}
close(IN);
close(BRIDGE);
close(OUT);
print "bridge count $brcount intersection count $count\n";

# So now we have a text file that looks like this:
# 100007 1 200044
# 200044 2 100007
# 100011 1 200102
# 200102 2 100011
# 100038 2 200102
# segment id, to or from flag, and segment it intersects with -id
# next, put each in numerical order
# going back to arc/info to do this - much faster sort that way

# the text file with the necessary data is xxmult2.txt
# arc will need to use this filename
system("rm xxunload.txt");

```

### B.11.4 154bridges3.perl

```

#! /usr/local/bin/perl

# 154bridges3.perl
# -----
#   Program: 154bridges3.perl
#   Purpose: just an extension of 154bridges2.perl
# -----
#   Called By: 154bridges.aml
#   Calls Made: none
# -----
#   History: date - 27 August 1997 - Tobi Sellekaerts
#             steinbet@email.grafenwoehr.army.mil
#             http://199.56.131.202/~tobi/
#             Terrain Simulations - 7th ATC - Grafenwoehr, Germany
# -----

print "*****Adding intersection information to the road &
river files\n";
# now actually going through and adding the required text to the
# river and road output files

# first, separate them out to different files based on road or river
open(IN, "xxunload.txt");
open(OUTRD, ">xxrdint.txt");
open(OUTHY, ">xxhyint.txt");

while($in = <IN>) {
    if(substr($in, 3, 1) == 1) {
        print OUTRD "$in";
    } else {
        print OUTHY "$in";
    }
}

close(IN);
close(OUTRD);
close(OUTHY);

# second, cycle through each of the text files and write the
# appropriate wording out
# now the text file looks like this:
# 100001 1 100148
# 100002 1 100003
# 100003 1 100002
# 100003 2 100235
# 100004 1 100063
# 100004 2 100003
# Slightly different spacing; same information

# reading out the beginning numbers of each file first
open(STATS, "xxrdstat.txt");      # working on roads first
$in = <STATS>;      # the first one will always be 100001
$in = <STATS>;      # the id at the beginning of the second file
chop($in);

```

```

$rdbegin2 = $in * 1;
$in = <STATS>;      # the id at the beginning of the third file
chop($in);
$rdbegin3 = $in * 1;
close(STATS);

open(STATS2, "xxhydstat.txt");      # working on hydro second
$in = <STATS2>;      # the first one will always be 200001
$in = <STATS2>;      # the id at the beginning of the second file
chop($in);
$hybegin2 = $in * 1;
$in = <STATS2>;      # the id at the beginning of the third file
chop($in);
$hybegin3 = $in * 1;
close(STATS2);

# =====
# Working on rivers first
# =====
# =====

# hydro file number one
open(INT, "xxhyint.txt");
open(IN, "xxhydroout1.txt");
open(OUT, ">xxtemp.txt");
open(DUPS, ">xxdups.txt");
$duptot = 0;

$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$sft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;

print "Working on xxhydroout1.txt...";
# we only loop as long as the file has entries
for($i = 200001; $i < $hybegin2; $i++) {
# for($i = 200001; $i < 200003; $i++) {
    $in = <IN>;

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;
        $length = $endpt - 4;
        $verts = substr($in, 4, $length) * 1;
        print OUT "$in";      # output first header line
        $in = <IN>;
        print OUT "$in";      # output second header line

        # determine if it is intersecting with a road or river
        if($merge > 200000) {
            $mergetype = "river";
        } else {
            $mergetype = "road";
        }
    }
}

```

```

# if $ft is 1, it is a FROM node, $ft = 2 is a TO node
if($ft == 2) {
    for($k = 1; $k < $verts; $k++) {
        $in = <IN>;
        print OUT "$in";
    }
}
$in = <IN>;          # this is the line that needs the mod
chop($in);

if($ft == 2) {
    print OUT "$in"      Intersection with $mergetype $merge\n";
} else {
    print OUT "$in" Intersection with $mergetype $merge\n";
}

# cycle through until we're at the end of the arc to write
# out the rest of the arc to the output file
$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}
$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$sft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUPS "$int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $sft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}

```

```

        }
    }
print OUT "END\n";
close(IN);
close(OUT);
system("mv xxtemp.txt xxhydroout1.txt");

print "xxhydroout2.txt...";
open(IN, "xxhydroout2.txt");
open(OUT, ">xxtemp.txt");

# we only loop as long as the file has entries
for($i = $hybegin2; $i < $hybegin3; $i++) {
# for($i = 200001; $i < 200003; $i++) {
    $in = <IN>

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;
        $length = $endpt - 4;
        $verts = substr($in, 4, $length) * 1;
        print OUT "$in";      # output first header line
        $in = <IN>;
        print OUT "$in";      # output second header line

        # determine if it is intersecting with a road or river
        if($merge > 200000) {
            $mergetype = "river";
        } else {
            $mergetype = "road";
        }

        # if $ft is 1, it is a FROM node, $ft = 2 is a TO node
        if($ft == 2) {
            for($k = 1; $k < $verts; $k++) {
                $in = <IN>;
                print OUT "$in";
            }
        }

        $in = <IN>;           # this is the line that needs the mod
        chop($in);

        if($ft == 2) {
            print OUT "$in      Intersection with $mergetype $merge\n";
        } else {
            print OUT "$in Intersection with $mergetype $merge\n";
        }
        # cycle through until we're at the end of the arc to write
        # out the rest of the arc to the output file
        $a = 1;
        while($a) {
            $in = <IN>;
            print OUT "$in";
    }
}

```

```

        if(substr($in, 0, 3) eq "END") {
            $a = 0;
        }
    }

    $previd = $rivid;

    # read another 'arc to be processed' number out of the
    intersections file
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $ft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
    while($rivid == $previd) {
        print DUPS "$int";
        ++$duptot;
        $int = <INT>;
        $rivid = substr($int, 0, 9) * 1;
        $ft = substr($int, 9, 2) * 1;
        $merge = substr($int, 11, 9) * 1;
    }

} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}

print OUT "END\n";
close(IN);
close(OUT);
system("mv xxtemp.txt xxhydroout2.txt"); # uncomment this later!

print "xxhydroout3.txt\n";
open(IN, "xxhydroout3.txt");
open(OUT, ">xxtemp.txt");

$i = $hybegin3;

# we only loop as long as the file has entries
while($in = <IN>) {          # changing the loop for this last one
# for($i = $hybegin3; $i < $hybegin3; $i++) {

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;

```

```

$length = $endpt - 4;
$verts = substr($in, 4, $length) * 1;
print OUT "$in";      # output first header line
$in = <IN>;
print OUT "$in";      # output second header line

# determine if it is intersecting with a road or river
if($merge > 200000) {
    $mergetype = "river";
} else {
    $mergetype = "road";
}

# if $ft is 1, it is a FROM node, $ft = 2 is a TO node
if($ft == 2) {
    for($k = 1; $k < $verts; $k++) {
        $in = <IN>;
        print OUT "$in";
    }
}
$in = <IN>;           # this is the line that needs the mod
chop($in);

if($ft == 2) {
    print OUT "$in      Intersection with $mergetype $merge\n";
} else {
    print OUT "$in Intersection with $mergetype $merge\n";
}

# cycle through until we're at the end of the arc to write
# out the rest of the arc to the output file
$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}
$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$int = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUPS "$int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $int = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

```

```

} elsif(substr($in, 0, 3) eq "END") {
    print OUT "$in";
} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}
close(IN);
close(OUT);
system("mv xxtemp.txt xxhydroout3.txt");

# =====
# =====
# Now repeat the whole thing for duplicate items
# =====
# =====

# copy the duplicates file and start over with an empty one
system("mv xxdups.txt xxhyint.txt");

while($duptot > 0) {
    print "Number of duplicates is $duptot\n";
    $duptot = 0;
    # hydro file number one
    open(INT, "xxhyint.txt");
    open(IN, "xxhydroout1.txt");
    open(OUT, ">xxtemp.txt");
    open(DUPS, ">xxdups.txt");

    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $ft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;

    print "Working on duplicates for xxhydroout1.txt...";
    # we only loop as long as the file has entries
    for($i = 200001; $i < $hybegin2; $i++) {
        $in = <IN>

        # if we happen to be on an arc that gets a comment, process it
        if($i == $rivid) {

            # find the number of vertices
            $endpt = index($in, " ", 5) * 1;
            $length = $endpt - 4;
            $verts = substr($in, 4, $length) * 1;
            print OUT "$in";      # output first header line
        }
    }
}

```

```

$in = <IN>;
print OUT "$in";      # output second header line

# determine if it is intersecting with a road or river
if($merge > 200000) {
    $mergetype = "river";
} else {
    $mergetype = "road";
}

# if $ft is 1, it is a FROM node, $ft = 2 is a TO node
if($ft == 2) {
    for($k = 1; $k < $verts; $k++) {
        $in = <IN>;
        print OUT "$in";
    }
}

$in = <IN>;           # this is the line that needs the mod
chop($in);

# determine whether or not there is already info there
if(substr($in, 37, 1) eq "I") {
    print OUT "$in, $mergetype $merge\n";
} else {
    if($ft == 2) {
        print OUT "$in      Intersection with $mergetype
$merge\n";
    } else {
        print OUT "$in Intersection with $mergetype $merge\n";
    }
}

# cycle through until we're at the end of the arc to write
# out the rest of the arc to the output file
$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}

$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUPS "$int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
}

```

```

$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
}

} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}
print OUT "END\n";
close(IN);
close(OUT);
system("mv xxtemp.txt xxhydroout1.txt");

print "xxhydroout2.txt...";
open(IN, "xxhydroout2.txt");
open(OUT, ">xxtemp.txt");

# we only loop as long as the file has entries
for($i = $hybegin2; $i < $hybegin3; $i++) {
    $in = <IN>

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;
        $length = $endpt - 4;
        $verts = substr($in, 4, $length) * 1;
        print OUT "$in";      # output first header line
        $in = <IN>;
        print OUT "$in";      # output second header line

        # determine if it is intersecting with a road or river
        if($merge > 200000) {
            $mergetype = "river";
        } else {
            $mergetype = "road";
        }

        # if $ft is 1, it is a FROM node, $ft = 2 is a TO node
        if($ft == 2) {
            for($k = 1; $k < $verts; $k++) {
                $in = <IN>;
                print OUT "$in";
            }
        }
    }
}

```

```

$in = <IN>;           # this is the line that needs the mod
chop($in);

# determine whether or not there is already info there
if(substr($in, 37, 3) eq "Int") {
    print OUT "$in, $mergetype $merge\n";
} else {
    if($ft == 2) {
        print OUT "$in      Intersection with $mergetype
$merge\n";
    } else {
        print OUT "$in Intersection with $mergetype $merge\n";
    }
}

# cycle through until we're at the end of the arc to write
# out the rest of the arc to the output file
$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}

$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUP$ "$int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $ft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}

```

```

    }
}
print OUT "END\n";
close(IN);
close(OUT);
system("mv xxtemp.txt xxhydroout2.txt"); # uncomment this later!

print "xxhydroout3.txt\n";
open(IN, "xxhydroout3.txt");
open(OUT, ">xxtemp.txt");

$in = $hybegin3;

# we only loop as long as the file has entries
while($in = <IN>) {          # changing the loop for this last one

    # if we happen to be on an arc that gets a comment, process it
    if($in == $rivid) {

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;
        $length = $endpt - 4;
        $verts = substr($in, 4, $length) * 1;
        print OUT "$in";      # output first header line
        $in = <IN>;
        print OUT "$in";      # output second header line

        # determine if it is intersecting with a road or river
        if($merge > 200000) {
            $mergetype = "river";
        } else {
            $mergetype = "road";
        }

        # if $ft is 1, it is a FROM node, $ft = 2 is a TO node
        if($ft == 2) {
            for($k = 1; $k < $verts; $k++) {
                $in = <IN>;
                print OUT "$in";
            }
        }
        $in = <IN>;           # this is the line that needs the mod
        chop($in);

        # determine whether or not there is already info there
        if(substr($in, 37, 3) eq "Int") {
            print OUT "$in, $mergetype $merge\n";
        } else {
            if($ft == 2) {
                print OUT "$in      Intersection with $mergetype
$merge\n";
            } else {
                print OUT "$in Intersection with $mergetype $merge\n";
            }
        }
    }

    # cycle through until we're at the end of the arc to write
}

```

```

# out the rest of the arc to the output file
$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}

$previd = $rivid;

# read another 'arc to be processed' number out of the
# intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUPS "$int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $ft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

} elsif(substr($in, 0, 3) eq "END") {
    print OUT "$in";
} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}
close(IN);
close(OUT);
system("mv xxtemp.txt xxhydroout3.txt");

close(INT);
close(DUPS);
system("mv xxdups.txt xxhyint.txt");
print "$duptot duplicates remain\n";
}

# =====
# =====
# now repeat the entire thing for roads
# =====
# =====

# trans file number one
open(INT, "xxrdint.txt");
open(IN, "xxtransout1.txt");
open(OUT, ">xxtemp.txt");
open(DUPS, ">xxdups.txt");
$duptot = 0;

$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;

print "Working on xxtransout1.txt";
# we only loop as long as the file has entries
for($i = 100001; $i < $rdbegin2; $i++) {
    $in = <IN>

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;
        $length = $endpt - 4;
        $verts = substr($in, 4, $length) * 1;
        print OUT "$in";      # output first header line
        $in = <IN>;
        print OUT "$in";      # output second header line

        # determine if it is intersecting with a road or river
        if($merge > 200000) {
            $mergetype = "river";
        } else {
            $mergetype = "road";
        }

        # if $ft is 1, it is a FROM node, $ft = 2 is a TO node
        if($ft == 2) {
            for($k = 1; $k < $verts; $k++) {
                $in = <IN>;
                print OUT "$in";
            }
            $in = <IN>;          # this is the line that needs the mod
            chop($in);

            if($ft == 2) {
                print OUT "$in      Intersection with $mergetype $merge\n";
            } else {
                print OUT "$in Intersection with $mergetype $merge\n";
            }
        }

        # cycle through until we're at the end of the arc to write
        # out the rest of the arc to the output file
    }
}

```

```

$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}

$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$sft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUP$ "int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $sft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}
print OUT "END\n";
close(IN);
close(OUT);
system("mv xxtemp.txt xxtransout1.txt");

print "...xxtransout2.txt...";
open(IN, "xxtransout2.txt");
open(OUT, ">xxtemp.txt");

# we only loop as long as the file has entries
for($i = $rdbegin2; $i < $rdbegin3; $i++) {
    $in = <IN>

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

```

```

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;
        $length = $endpt - 4;
        $verts = substr($in, 4, $length) * 1;
        print OUT "$in";      # output first header line
        $in = <IN>;
        print OUT "$in";      # output second header line

        # determine if it is intersecting with a road or river
        if($merge > 200000) {
            $mergetype = "river";
        } else {
            $mergetype = "road";
        }

        # if $ft is 1, it is a FROM node, $ft = 2 is a TO node
        if($ft == 2) {
            for($k = 1; $k < $verts; $k++) {
                $in = <IN>;
                print OUT "$in";
            }
        }

        $in = <IN>;           # this is the line that needs the mod
        chop($in);

        if($ft == 2) {
            print OUT "$in      Intersection with $mergetype $merge\n";
        } else {
            print OUT "$in Intersection with $mergetype $merge\n";
        }
        # cycle through until we're at the end of the arc to write
        # out the rest of the arc to the output file
        $a = 1;
        while($a) {
            $in = <IN>;
            print OUT "$in";
            if(substr($in, 0, 3) eq "END") {
                $a = 0;
            }
        }
}

$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$sft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUP$ "int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $sft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

```

```

    }

} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}

print OUT "END\n";
close(IN);
close(OUT);
system("mv xxtemp.txt xxtransout2.txt"); # uncomment this later!

print "xxtransout3.txt\n";
open(IN, "xxtransout3.txt");
open(OUT, ">xxtemp.txt");

$i = $rdbegin3;

# we only loop as long as the file has entries
while($in = <IN>) {           # changing the loop for this last one
    # for($i = $rdbegin3; $i < $rdbegin3; $i++) {

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;
        $length = $endpt - 4;
        $verts = substr($in, 4, $length) * 1;
        print OUT "$in";      # output first header line
        $in = <IN>;
        print OUT "$in";      # output second header line

        # determine if it is intersecting with a road or river
        if($merge > 200000) {
            $mergetype = "river";
        } else {
            $mergetype = "road";
        }

        # if $ft is 1, it is a FROM node, $ft = 2 is a TO node
        if($ft == 2) {
            for($k = 1; $k < $verts; $k++) {
                $in = <IN>;
                print OUT "$in";
            }
        }
    }
}

```

```

$in = <IN>;                  # this is the line that needs the mod
chop($in);

if($ft == 2) {
    print OUT "$in      Intersection with $mergetype $merge\n";
} else {
    print OUT "$in Intersection with $mergetype $merge\n";
}

# cycle through until we're at the end of the arc to write
# out the rest of the arc to the output file
$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}

$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUP "$int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $ft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

} elsif(substr($in, 0, 3) eq "END") {
    print OUT "$in";
} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}
close(IN);
close(OUT);
system("mv xxtemp.txt xxtransout3.txt");

```

```

# =====
# Now repeat the whole thing for duplicate items
# =====
# copy the duplicates file and start over with an empty one
system("mv xxdups.txt xxrdint.txt");

while($duptot > 0) {
    print "Number of duplicates is $duptot\n";
    $duptot = 0;
    # trans file number one
    open(INT, "xxrdint.txt");
    open(IN, "xxtransout1.txt");
    open(OUT, ">xxtemp.txt");
    open(DUPS, ">xxdups.txt");

    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $ft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;

    print "Working on duplicates for xxtransout1.txt...";
    # we only loop as long as the file has entries
    for($i = 100001; $i < $rdbegin2; $i++) {
        $in = <IN>;

        # if we happen to be on an arc that gets a comment, process it
        if($i == $rivid) {

            # find the number of vertices
            $endpt = index($in, " ", 5) * 1;
            $length = $endpt - 4;
            $verts = substr($in, 4, $length) * 1;
            print OUT "$in";      # output first header line
            $in = <IN>;
            print OUT "$in";      # output second header line

            # determine if it is intersecting with a road or river
            if($merge > 200000) {
                $mergetype = "river";
            } else {
                $mergetype = "road";
            }

            # if $ft is 1, it is a FROM node, $ft = 2 is a TO node
            if($ft == 2) {
                for($k = 1; $k < $verts; $k++) {
                    $in = <IN>;
                    print OUT "$in";
                }
            }

            $in = <IN>;          # this is the line that needs the mod
            chop($in);
        }
    }
}

```

```

# determine whether or not there is already info there
if(substr($in, 37, 1) eq "I") {
    print OUT "$in, $mergetype $merge\n";
} else {
    if($ft == 2) {
        print OUT "$in      Intersection with $mergetype
$merge\n";
    } else {
        print OUT "$in Intersection with $mergetype $merge\n";
    }
}

# cycle through until we're at the end of the arc to write
# out the rest of the arc to the output file
$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}

$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUPS "$int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $ft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}

print OUT "END\n";

```

```

close(IN);
close(OUT);
system("mv xxtemp.txt xxtransout1.txt");

print "xxtransout2.txt...";
open(IN, "xxtransout2.txt");
open(OUT, ">xxtemp.txt");

# we only loop as long as the file has entries
for($i = $rdbegin2; $i < $rdbegin3; $i++) {
    $in = <IN>

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

        # find the number of vertices
        $endpt = index($in, " ", 5) * 1;
        $length = $endpt - 4;
        $verts = substr($in, 4, $length) * 1;
        print OUT "$in";      # output first header line
        $in = <IN>;
        print OUT "$in";      # output second header line

        # determine if it is intersecting with a road or river
        if($merge > 200000) {
            $mergetype = "river";
        } else {
            $mergetype = "road";
        }

        # if $ft is 1, it is a FROM node, $ft = 2 is a TO node
        if($ft == 2) {
            for($k = 1; $k < $verts; $k++) {
                $in = <IN>;
                print OUT "$in";
            }
        }
        $in = <IN>;          # this is the line that needs the mod
        chop($in);

        # determine whether or not there is already info there
        if(substr($in, 37, 3) eq "Int") {
            print OUT "$in, $mergetype $merge\n";
        } else {
            if($ft == 2) {
                print OUT "$in      Intersection with $mergetype
$merge\n";
            } else {
                print OUT "$in Intersection with $mergetype $merge\n";
            }
        }

        # cycle through until we're at the end of the arc to write
        # out the rest of the arc to the output file
        $a = 1;
        while($a) {
    }
}

```

```

$in = <IN>;
print OUT "$in";
if(substr($in, 0, 3) eq "END") {
    $a = 0;
}
}

$rivid = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $prevrid) {
    print DUPS "$int";
    ++$duptot;
    $int = <INT>;
    $rivid = substr($int, 0, 9) * 1;
    $ft = substr($int, 9, 2) * 1;
    $merge = substr($int, 11, 9) * 1;
}

} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}
print OUT "END\n";
close(IN);
close(OUT);
system("mv xxtemp.txt xxtransout2.txt"); # uncomment this later!

print "xxtransout3.txt\n";
open(IN, "xxtransout3.txt");
open(OUT, ">xxtemp.txt");

$i = $rdbegin3;

# we only loop as long as the file has entries
while($in = <IN>) {           # changing the loop for this last one

    # if we happen to be on an arc that gets a comment, process it
    if($i == $rivid) {

        # find the number of vertices
    }
}

```

```

$endpt = index($in, " ", 5) * 1;
$length = $endpt - 4;
$verts = substr($in, 4, $length) * 1;
print OUT "$in";      # output first header line
$in = <IN>;
print OUT "$in";      # output second header line

# determine if it is intersecting with a road or river
if($merge > 200000) {
    $mergetype = "river";
} else {
    $mergetype = "road";
}

# if $ft is 1, it is a FROM node, $ft = 2 is a TO node
if($ft == 2) {
    for($k = 1; $k < $verts; $k++) {
        $in = <IN>;
        print OUT "$in";
    }
}
$in = <IN>;           # this is the line that needs the mod
chop($in);

# determine whether or not there is already info there
if(substr($in, 37, 3) eq "Int") {
    print OUT "$in, $mergetype $merge\n";
} else {
    if($ft == 2) {
        print OUT "$in      Intersection with $mergetype
$merge\n";
    } else {
        print OUT "$in Intersection with $mergetype $merge\n";
    }
}

# cycle through until we're at the end of the arc to write
# out the rest of the arc to the output file
$a = 1;
while($a) {
    $in = <IN>;
    print OUT "$in";
    if(substr($in, 0, 3) eq "END") {
        $a = 0;
    }
}

$previd = $rivid;

# read another 'arc to be processed' number out of the
intersections file
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
while($rivid == $previd) {
    print DUPS "$int";
}

```

```

++$duptot;
$int = <INT>;
$rivid = substr($int, 0, 9) * 1;
$ft = substr($int, 9, 2) * 1;
$merge = substr($int, 11, 9) * 1;
}

} elsif(substr($in, 0, 3) eq "END") {
    print OUT "$in";
} else {
    # if we're not on an arc that gets processed, skip it
    print OUT "$in";
    $c = 1;

    # cycle through until we're at the end of the arc
    while($c) {
        $in = <IN>;
        print OUT "$in";
        if(substr($in, 0, 3) eq "END") {
            $c = 0;
        }
    }
}
close(IN);
close(OUT);
system("mv xxtemp.txt xxtransout3.txt");

close(INT);
close(DUPS);
system("mv xxdupts.txt xxrdint.txt");
print "$duptot duplicates remain\n";
}

# finally! finished the previous section on 28 August 1997

# =====
# =====
# second half of the program - creating the bridges output file
# =====
# =====

print *****Creating the bridges output
file\n";
# We have a text file, xxbridges.txt, that looks like this:
# 18 100007 1 200044 2
# 133 100011 1 200102 2
# 133 100038 2 200102 2
# nodeid roadidroad_type riverid river_type

# ...and xxnodexy.txt (pretty self explanatory)
#     1          24.73925      12.00144
#     2          14.19168      12.00143
#     3          24.82953      12.00140

# We need to turn it into a CBS bridges file

```

```

# Each bridge/ford is written to the file as a set 4 ASCII records as
follows:
# Record: 1 Brdg_Type 1 2 Brdg_ID Road_ID Road_Type River_ID
River_Type
# Record: 2 Brdg_Orientation(deg) 0.0 0.0 0.0 0.0
# Record: 3 Long(deg) Lat(deg) Military_Load_Class(real)
# Record: 4 END

# Two examples:
# 701 1 2 700001 100002 1 200002 1
# 14.211990 0.0 0.0 0.0 0.0
# 10.549685 54.988102 10.000000
# END
# 701 1 2 700002 100004 1 200011 1
# 158.519958 0.0 0.0 0.0 0.0
# 9.156546 54.974953 10.000000

open(OUT, ">xxbridges2.txt");
open(NODE, "xxnodexy.txt");
open(BRIDGE, "xxbridges.txt");

$testbridge = 0;
$begincount = 1;      # counting our way through the nodexy file

# work through the xxbridges.txt file, processing each bridge
individually
# all we're doing on this loop is adding the x and y information to each
bridge
while($bridge = <BRIDGE>) {
    chop($bridge);
    $spl = index($bridge, " ", 1);
    $nodeno = substr($bridge, 0, $spl) * 1;

    # if this bridge isn't a duplicate of the one before it, then process
    it
    if($nodeno != $testbridge) {
        $testbridge = $nodeno;

        # find the x and y values of this particular node; from
        xxnodexy.txt
        # first, cycle through the nodexy file to find the correct line
        for($i = $begincount; $i <= $nodeno; ++$i) {
            $nodexy = <NODE>;
        }
        $begincount = $nodeno;

        # second, assign the actual x and y to variables
        $nodex = substr($nodexy, 8, 16) * 1;
        $nodey = substr($nodexy, 25, 16) * 1;

        # write all the necessary data to the output file
        print OUT "$bridge $nodex $nodey\n";
    }
}

close(OUT);
close(NODE);
close(BRIDGE);

# Now, headed back out to arc to put these things in order
# and find the angle of each bridge
system("rm xxbridges3.txt");
system("rm xxangle.txt");

```

### B.11.5 154bridges4.perl

```
#!/usr/local/bin/perl
# 154bridges4.perl
#
#      Program: 154bridges4.perl
#      Purpose: just an extension of 154bridges3.perl
#
#      Called By: 154bridges.aml
#      Calls Made: none
#
#      History: date - 27 August 1997 - Tobi Sellekaerts
#                  steinbet@email.grafenwoehr.army.mil
#                  http://199.56.131.202/~tobi/
#                  Terrain Simulations - 7th ATC - Grafenwoehr, Germany
#
open(BRIDGE, "xxbridges3.txt");
open(OUT, ">final_bridges.txt");
open(ANGLE, "xxangle.txt");# bridges are feature #7
# each bridge will be numbered sequentially
$count = 700001;

# work through the xxbridges.txt file, processing each bridge
individually
while($bridge = <BRIDGE>) {
    $nodeno = substr($bridge, 0, 9) * 1;
    # find the number of the river and road for this bridge
    # this is also coming out of xxbridge.txt
    $road = substr($bridge, 9, 9) * 1;
    $river = substr($bridge,20, 9) * 1;
    $rdtype = substr($bridge, 18, 2) * 1;
    $rivtype = substr($bridge, 29, 2) * 1;
    # second, assign the actual x and y to variables
    $nodex = substr($bridge, 31, 10) * 1;
    $nodey = substr($bridge, 41, 10) * 1;
    # find the angle of the bridge at that point on the road
    # this is coming from xxangles.txt
    $angle = <ANGLE>;
    $angle = <ANGLE>;
    chop($angle);
    $angle = $angle * 1;
    # write all the necessary data to the output file
    print OUT " 701 1 2 $count $road $rdtype $river $rivtype\n";
    printf OUT "%11.6f 0.0 0.0 0.0 0.0\n", $angle;
    printf OUT "%11.6f %11.6f %11.6f\n", $nodex, $nodey, 10;
    print OUT "END\n";
    ++$count;
}
print OUT "END\n";
close(ANGLE);
close(OUT);
close(BRIDGE);
close(NODE);
```

### B.12 154others.perl

```
#!/usr/local/bin/perl
#
#      Program: 154others.perl
#      Purpose: to create all those other extra files that don't
#              really fit anywhere else. lines, parms, map, rad
#
#      Called By: dcwcbs154.aml
#      Calls Made: none
#
#      Arguments: none
#      Globals: none
#      Locals:
#
#      History: 3 September 1997 - Tobi Sellekaerts
#                  steinbet@email.grafenwoehr.army.mil
#                  http://199.56.131.202/~tobi/
#                  Terrain Simulations - 7th ATC - Grafenwoehr, Germany
#
# first working on the lines files
open(OUT, ">xxline0.dat");
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
close(OUT);

open(OUT, ">xxline1.dat");
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
close(OUT);

open(OUT, ">xxline2.dat");
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
close(OUT);

open(OUT, ">xxlines.dat");
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
close(OUT);
```

```

# next working on the rad files
open(OUT, ">xxrad0.dat");
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
close(OUT);

open(OUT, ">xxrad1.dat");
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
close(OUT);

open(OUT, ">xxrad2.dat");
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
print OUT "99999 99 99 9999999 9999999 9999999 9999999 9999 0
0 Sentinel\n";
close(OUT);

# and that's all folks...

```

## Appendix C: Points of contact

### Terrain Simulation (TerraSim), a Logicon lab Dave Knox, TerraSim manager

World Wide Web Homepage	<a href="http://usa.7atc.army.mil/">http://usa.7atc.army.mil/</a>
Telephone Number	DSN 474-3111
Fax Number	Civilian 011 49 9641 833111
U.S. Mailing Address	Civilian 011 49 9641 550 TerraSim attn: Dave Knox Logicon TSI Unit 28130 APO AE 09114
Overseas Mailing Address	TerraSim attn: Dave Knox Gebaeude 1460 Camp Aachen Truppenuebungplatz 92655 Grafenwoehr, Germany

Paper Author (can also be reached at the above numbers and addresses)

Tobi L.S. Sellekaerts	<a href="mailto:steinbet@email.grafenwoehr.army.mil">steinbet@email.grafenwoehr.army.mil</a>
-----------------------	--

W. Bruce Bollinger, the SCOR for TerraSim at USAREUR 7th Army Training Command

Telephone Number	DSN 474-2396 or 2460
Fax Number	Civilian 011 49 9641 83 2396 or 2460 DSN 474-2541
U.S. Mailing Address	Civilian 011 49 9641 832541 HQ 7th ATC Attn: AEAGC-TS-F Unit 28130 APO AE 09114
Email Address	<a href="mailto:bollingb@hq.7atc.army.mil">bollingb@hq.7atc.army.mil</a>